

# CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

4 de setembro de 2020

## Resumo

Tipos Abstratos de Dados: introdução

# Objetivos da aula

- Apresentar os tipos abstratos de dados
- Entender a abstração do dado

- Definição do conceito de TAD
- Resolução de um problema em alto nível usando o TAD
- Não conheceremos a implementação do TAD

- Um Tipo Abstrato de Dados (TAD) é basicamente abstrair o dado que está em memória pela definição de operações abstratas que se preocupam fundamentalmente em *o que* fazer e não em *como* fazer.

- Operações sobre os dados:
  - protótipos das funções e procedimentos
  - conhecimento do que elas fazem, não de como elas fazem
  - implementação irrelevante no momento
- imagine que o compilador te fornece o tipo e as implementações das funções e procedimentos

- Vamos apresentar um TAD básico: o TAD *Conjunto*
- A forma de apresentação será:
  - Definição do conceito de conjunto;
  - Apresentação de um problema que podem ser resolvidos com este conceito;
  - Apresentação de uma solução em alto nível para este problema.

- Um conjunto é um conceito matemático bem conhecido
- É uma coleção de elementos sem repetição
- Operações importantes sobre conjuntos:
  - saber se o conjunto é vazio
  - saber se um elemento pertence ou não a um conjunto
  - realizar operações de união, intersecção e diferença
  - ...

# O TAD conjunto

- Um conjunto será constituído de números inteiros sem repetições
- Conhecemos os protótipos das funções e procedimentos
- Considere a existência de um tipo *conjunto* e uma variável deste tipo:

```
var c: conjunto;
```



# Operações sobre o TAD conjunto

```
function conjunto_vazio (c: conjunto): boolean;  
// Retorna true se o conjunto c eh vaziao e false caso contrario.  
  
function cardinalidade (c: conjunto): longint;  
// Retorna a cardinalidade do conjunto c.  
  
procedure inserir_conjunto (x: longint; var c: conjunto);  
// Insere o elemento x no conjunto c.  
  
procedure remover_conjunto (x: longint; var c: conjunto);  
// Remove o elemento x do conjunto c.  
  
function uniao (c1, c2: conjunto): conjunto;  
// Obtem a uniao dos conjuntos c1 e c2.  
  
function interseccao (c1, c2: conjunto): conjunto;  
// Obtem a interseccao dos conjuntos c1 e c2.  
  
function diferenca (c1, c2: conjunto): conjunto  
// Obtem a diferenca dos conjuntos c1 e c2 (c1 - c2).
```

# Operações sobre o TAD conjunto

```
function pertence (x: longint; c: conjunto): boolean;  
// Retorna true se x pertence ao conjunto c e false caso contrario.
```

```
function sao_iguais (c1, c2: conjunto): boolean;  
// Retorna true se o conjunto c1 = c2 e false caso contrario.
```

```
function contido (c1, c2: conjunto): boolean;  
// Retorna true se o conjunto c1 esta contido no conjunto c2 e false caso contr
```

```
function copiar_conjunto (c1: conjunto): conjunto;  
// Copia os elementos do conjunto c1 para outro conjunto.
```

# Operações especiais sobre o TAD conjunto

- Estas operações são especiais:

```
procedure inicializar_conjunto (var c: conjunto);  
// Cria um conjunto vazio.  
// Deve ser chamado antes de qualquer operacao no conjunto.  
  
function retirar_um_elemento (var c: conjunto): longint;  
// Escolhe um elemento qualquer do conjunto para ser removido  
// remove, e o retorna.  
  
procedure iniciar_proximo (var c: conjunto);  
// Inicializa o contador que sera usado na funcao incrementar_proximo.  
  
function incrementar_proximo (var c: conjunto; var x: longint): boolean;  
// Incrementa o contador e retorna x por referencia.  
// Retorna false se acabou conjunto
```

# Operações sobre o TAD conjunto

- *inicializar\_conjunto*: serve para inicializar a estrutura do conjunto e sua chamada deve anteceder qualquer outra;

# Operações sobre o TAD conjunto

- *retirar\_um\_elemento*: útil quando se deseja fazer uma determinada operação sobre todos os elementos do conjunto
- Normalmente é utilizada em um laço junto da função *conjunto\_vazio*
- Exemplo:

```
1 // trecho de código para imprimir todos os elementos.  
2 // o efeito é que o conjunto c estará vazio ao final.  
3   while not conjunto_vazio (c) do  
4     begin  
5       x:= retirar_um_elemento (c);  
6       writeln (x);  
7     end;
```

# Operações sobre o TAD conjunto

- *iniciar\_proximo* e *incrementar\_proximo*: estas funções também permitem que se opere sobre todos os elementos do conjunto, mas sem removê-los.
- O sentido disso é mais ou menos o mesmo de quando queremos percorrer um vetor
- $i := 1, i := i + 1$ .

```
1 // outro trecho de código para imprimir todos os elementos.  
2 // o efeito é que o conjunto c estará intacto ao final.  
3   iniciar_proximo (c);  
4   while incrementar_proximo (c,x) do  
5     writeln (x);
```

- Problema que pode ser resolvido usando-se o TAD conjunto
- A Caixa Econômica Federal tem que descobrir duas vezes por semana se existem vencedores do concurso da megassena.
- A megassena é um concurso que paga um prêmio milionário para os apostadores que acertarem seis dezenas que são sorteadas. Cada apostador pode apostar de 6 a 15 dezenas. As dezenas são valores inteiros de 01 a 60.

# Entrada de dados

- O programa deve ler o sorteio feito em cerimônia pública pela Caixa Econômica Federal. Devemos fazer a carga de exatamente 6 números entre 01 e 60;
- Em seguida, devemos ler um número  $N$  contendo o total de apostadores;
- Na sequência, devemos ler  $N$  linhas de dados, cada uma com a aposta de um apostador. O primeiro valor da linha é o número  $M$  de apostas deste apostador, isto é, um número entre 6 e 15. O restante da linha são  $M$  números de 01 a 60;
- Finalmente, para cada aposta, devemos compará-la com o sorteio, gerando como saída as apostas que fizeram a megassena (acertou 6 dezenas), ou a quina (acertou 5 dezenas) ou a quadra (acertou quatro dezenas).
- Encerrar o programa.



# Como o TAD ajuda?

- as variáveis *sorteio* e *aposta* são conjuntos
- uma aposta vence se a cardinalidade da interseção com o sorteio é 6

# Leitura dos dados no conjunto

```
1 procedure ler (var c: conjunto; tam_aposta: longint);  
2 var i, num: longint;  
3 begin  
4     inicializar_conjunto (c);  
5     for i:= 1 to tam_aposta do  
6         begin  
7             read (num);  
8             inserir_conjunto (num,c);  
9         end;  
10 end;
```

```
1 program megasena;
2 // usa um TAD conjunto ainda nao definido
3 // usa as funcoes e procedimentos acima definidos
4 var sorteio, aposta, intersec: conjunto;
5     N, num_acertos: longint;
6 begin
7     ler (sorteio,6);
8     read (N);
9     for i:= 1 to N do
10    begin
11        read (tam_aposta);
12        ler (aposta, tam_aposta);
13        intersec:= interseccao (sorteio,aposta);
14        num_acertos:= cardinalidade (intersec);
15        if num_acertos = 6 then
16            writeln ('aposta ',i,' ganhou a megasena!')
17        else if num_acertos = 5 then
18            writeln ('aposta ',i,' ganhou a quina!')
19        else if num_acertos = 4 then
20            writeln ('aposta ',i,' ganhou a quadra!')
21        // else nao imprime nada, o acerto foi no maximo 3 e eh perdedora
22    end;
23 end.
```

- Observar o *uso* da procedure *intersecção* e da função *cardinalidade*. Se estas implementarem corretamente a definição dos conceitos, o programa funciona perfeitamente.
- É possível *usar* o TAD conjunto sem sequer conhecer sua estrutura. Os dados estão abstraídos.

- 1 Sem olhar a solução do livro:
  - Leia o enunciado do "problema da celebridade" e tente resolvê-lo usando o TAD conjunto
  - Leia o capítulo sobre o TAD pilha, seção 12.2
- 2 Resolva o problema da celebridade usando o TAD Pilha, seção 12.1

- este material está no livro no capítulo 12, seção 12.1

- Slides feitos em  $\text{\LaTeX}$  usando beamer
- Licença

*Creative Commons* Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.<http://creativecommons.org/licenses/by-nc-nd/2.5/br/>

Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.<http://creativecommons.org/licenses/by-nc-nd/2.5/br/>