SCC-223 Estruturas de Dados I

Tipo Abstrato de Dados - TAD

Profa. Elaine Parros Machado de Sousa

ICMC USP SÃO CARLOS

Aula baseada em material dos professores Adenilso Simão e Rudinei Goularte.

Grupo de Bases de Dados e Imagens
USP - São Carlos

Introdução

- **Algoritmo**
  - Pode ser visto como uma sequência de ações factíveis para a obtenção de uma solução para um determinado tipo de problema
- **Estrutura de Dados**
  - Organização de dados e operações (algoritmos) que podem ser aplicadas sobre esses dados para solução de problemas
- **Programas**
  - Formulações concretas de algoritmos, baseados em representações e estruturas específicas de dados, considerando a linguagem de programação utilizada

Tipo de Dado

- **Tipo de dado** (ou Tipo): caracteriza o conjunto de valores que uma variável (ou uma expressão ou uma constante) pode assumir
  - ou o valor que pode ser gerado por uma função
  - ex:
    - tipos primitivos simples (`int`, `float`, `char`,...)
    - tipos estruturados (`struct`)
  - um conceito de programação

Tipo de Dado

- Os tipos de dados primitivos variam de uma linguagem de programação para outra
- O suporte para construção de tipos estruturados a partir dos tipos primitivos também varia de linguagem para linguagem

Tipo Abstrato de Dado (TAD)

- **TAD:** "*modelo acompanhado de uma coleção de operações definidas sobre este modelo*"
  - conjunto de valores (**V**) associados a um conjunto de operações (**O**) sobre estes valores
- Ex: modelo (*V*,*O*)
  - ***V*:** $\mathbb{Z}$
  - ***O*:** {+, -, *, /, <, >, =, ...}

Tipo Abstrato de Dado (TAD)

Independente de implementação

um mesmo tipo abstrato pode ser implementado de diversas maneiras na mesma linguagem e em diferentes linguagens de programação

Tipo Abstrato de Dado (TAD)

Propriedade Fundamental do TAD:

Ocultamento de Informação

(*Information Hidding*)

- ✓ separação entre **conceito** e **implementação**
- ✓ **encapsulamento**

TAD – Ocultamento de Informação

- **Encapsulamento** de tipos de dados
  - pensar em termos das operações suportadas e não como são implementadas
  - não é necessário conhecer sua representação interna => **caixa preta**
  - não se preocupa com a eficiência de tempo e espaço => são questões de implementação
- mudanças na implementação das operações do TAD não exigem alterações no programa que o utiliza

Vantagens do Uso de TAD

- Lógica mais clara
- Abstração
- Reuso
- Manutenção
- Portabilidade

PRODUTIVIDADE

MUNDO REAL

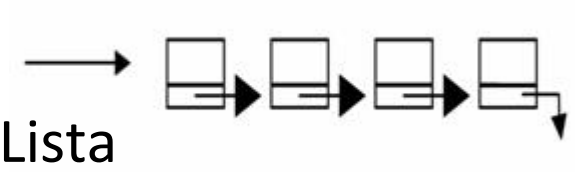
Cadastro de funcionários

Operações
-Inclui alguém no cadastro
-Tira alguém do cadastro
-Altera o cadastro
-Altera o salário

FUNCIONALIDADE

```
procedure  insere (X : elemento; var L : lista_ordenada);
   var p, anterior, newnode : lista_ordenada;
begin
p := l; anterior := nil;
while (p<> nil) and (p^.info < x) do
          begin
          anterior := p ;
          p := p^.next;
          end;
new( newnode );
newnode^.info := x;
newnode^.next := p;
If anterior = nil    {x será o primeiro da lista }
then L := newnode
else anterior^.next := newnode;
end;
```

IMPLEMENTAÇÃO



REPRESENTAÇÃO ABSTRATA

TAD - Exemplo

- TAD **RGB**
  - conjunto de valores (**V**): cores descritas segundo o sistema RGB
  - conjunto de operações (**O**): inserir cor, remover cor, compor duas cores, ...

Estrutura de dados:

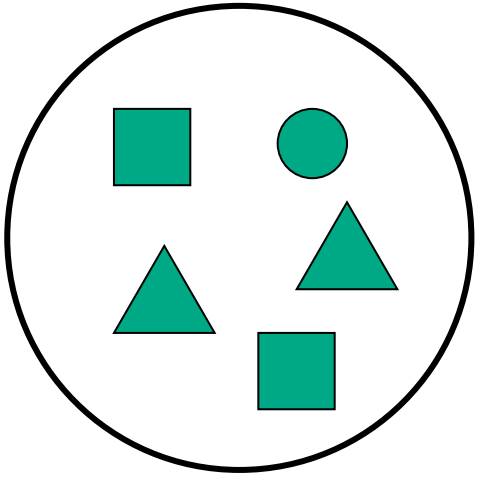
```
struct RGB {
        int R;
        int G;
        int B;
};
struct RGB cores;
```

OU

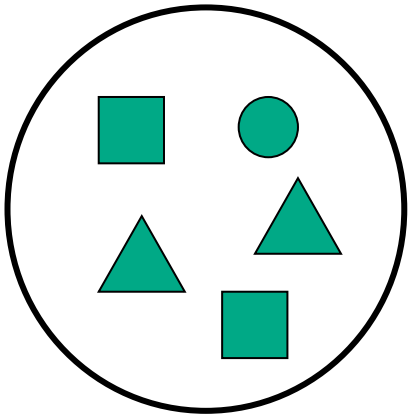
```
int cores[3];
```

TAD - Exemplo

- TAD **SACOLA (BAG)**
  - uma sacola (*bag*) pode conter zero ou mais elementos de um determinado tipo
  - os elementos podem aparecer múltiplas vezes
  - **Ex:** Sacola de Figuras Geométricas



TAD - Exemplo



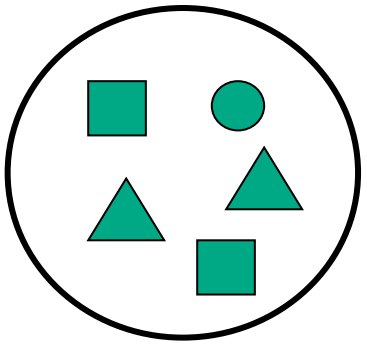
- TAD **SACOLA (BAG)**
  - Sacola de Figuras Geométricas
  - Operações:
    - Criar uma sacola vazia
    - Inserir um elemento na sacola
    - Consultar quantas ocorrências de um determinado elemento está presente na sacola
    - Remover um elemento da sacola
    - ....

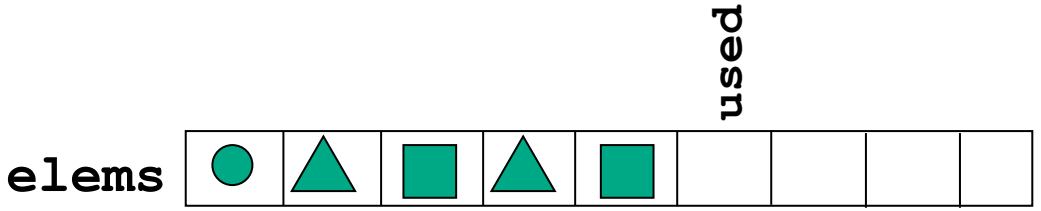
TAD Sacola

- Implementação
  - diversas maneiras de implementar um TAD
  - a escolha de uma estrutura de dados em particular deve levar em consideração características da aplicação
  - Ex:
    - volume de dados
    - funcionalidades
    - requisitos de espaço de armazenamento e/ou tempo de processamento
    - ...

TAD Sacola – VERSÃO 1

- Uma implementação "ingênua":
  - vetor **`elems`** com capacidade **`MaxElem`** de elementos do tipo **`ITEM`** da sacola
    - a limitação no número de elementos em uma sacola é restrição imposta pela **estratégia de implementação**
  - variável **`used`** que informa quantas posições de **`elems`** já foram utilizadas





TAD Sacola – VERSÃO 1

- Definição da estrutura (em pseudo-código)

```
Sacola {
        ITEM elems[MaxElem];
        inteiro used;
}
```

TAD Sacola – VERSÃO 1

- Operações (em pseudo-código - simplificado)
  - Criar sacola vazia

```
SacCriarVazia(){
      inicializa/aloca a sacola;
}
```

TAD Sacola – VERSÃO 1

- Operações (em pseudo-código – simplificado)
  - Inserir um elemento na sacola

```
SacInsereElem(ITEM e) {
      se (used < MaxElem)
      então
            elems[used]= e;
            used++;
      senão
        imprimir("A sacola esta cheia");
}
```

TAD Sacola – VERSÃO 1

- Operações (em pseudo-código - simplificado)
  - **Contagem**: quantas ocorrências há de um determinado elemento na sacola?

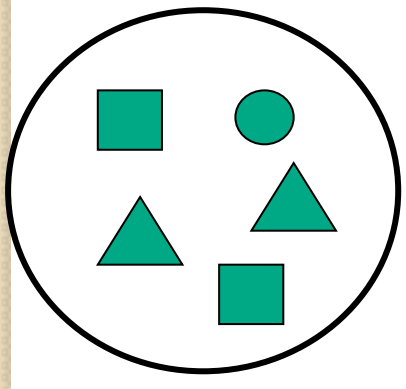
```
inteiro SacContaElem(ITEM e) {
   count = 0;
   para (index = 0; index < used; index++)
        se (elems[index] == e)
            count++;
    retorna count;
}
```

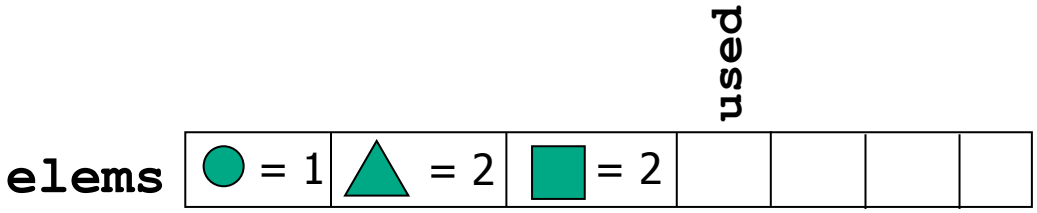
TAD Sacola – VERSÃO 1

- Pontos positivos da solução VERSÃO 1 para o TAD `Sacola`?
  - a inserção é relativamente simples
  - uma implementação aceitável se o número de inserções é (muito) maior que as demais operações
- Problemas?
  - mais espaço que necessário
  - custo da contagem é alto

TAD Sacola – VERSÃO 2

- Outra sugestão de implementação:
  - estrutura que armazena um par formado por um elemento do tipo **`ITEM`** e um inteiro que indicará o número de ocorrências
  - **`elems`** como um vetor de pares
  - variável **`used`** informa quantas posições de **`elems`** já foram utilizadas





TAD Sacola – Versão 2

- Definição da nova estrutura (em pseudo-código)

```
Elemento {
      ITEM valor;
      inteiro num;
};

Sacola {
      Elemento elems[MaxElem];
      inteiro used;
}
```

TAD Sacola – VERSÃO 2

- Operações (em pseudo-código - simplificado)
  - Criar sacola vazia
    - **mesma definição** de função mas com **implementação diferente**

```
SacCriarVazia(){
      inicializa/aloca a sacola;
}
```

TAD Sacola – Versão 2

- Operações (em pseudo-código - simplificado)
  - Inserir um elemento na sacola
    - **mesma definição** de função mas com **implementação diferente**

```
SacInsereElem(ITEM e) {

    ......
}
```

TAD Sacola – Versão 2

- Operações (em pseudo-código - simplificado)
  - **Contagem**: quantas ocorrências há de um determinado elemento na sacola?
    - **mesma definição** de função mas com **implementação diferente**

```
inteiro SacContaElem(ITEM e) {

      .......
}
```

TAD Sacola

- A mudança na implementação não afeta o modo como as funções são utilizadas
  - **ocultamento** de informações
  - mesmo TAD pode ser usado em diversas aplicações
  - alterações na implementação afetam o código das aplicações
- Outras melhorias poderiam ser feitas
  - manter o vetor ordenado segundo algum critério
  - usar uma árvore de busca...

TAD – Implementação em C

- Modularização em C
  - Módulo => arquivo fonte com extensão **.c**
    - **definição** da estrutura do TAD
    - **implementação** das funções do TAD
  - *Headers* => arquivos **.h** com a **interface** do TAD
    - definição do **tipo** TAD e "assinatura" das funções
  - Arquivo do programa principal – extenção **.c**
    - programa que usa o TAD: tipo e funções

TAD – Implementação em C

- Makefile – exemplos:

  - Mais simples: para compilar módulos e programa principal juntos:

```
all:
	gcc -std=c99 -Wall main.c sacola2.c -o sacolas
run:
	./sacolas;
```

TAD – Implementação em C

- Makefile – exemplos:
  - Para compilar tudo separadamente, geral código objeto (**.o**) e "linkar" na aplicação => mais adequado considerando a ideia do TAD

```
all: sacola2.o main.o
	gcc -std=c99 -Wall sacola2.o main.o -o sacolas

sacola2.o:
	gcc -c -std=c99 -Wall sacola2.c

main.o:
	gcc -c -std=c99 -Wall main.c

run:
	./sacolas;
```

Exercício para Entrega

Exercicio2_TAD.pdf