Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Roteiro 3

Rodrigo José Zonzin
212050002

1 TAD

1.1

Um Tipo Abstrato de Dados, ou TAD, é um modelo matemático que define uma estrutura regular de dados e as funções que operam sobre essa estrutura. Geralmente, essa modelagem tem como propósito permitir uma representação computacional de um objeto de interesse.

TADs permitem encapsulamento de código, legibilidade, coesão e um formalismo tão bom quanto necessário para se representar alguma estrutura de interesse.

Caso uma função que opere sobre os dados precise ser modificada - mas mantenha o mesmo comportamento - não é necessário alterar todo o código. Apenas um aquele bloco de código precisa ser alterado.

1.2

Por definição, um cubo é um paralelepípedo cujas base, altura e profundidade são iguais. Dessa maneira, precisamos de um único elemento inteiro para representar o "lado"do cubo.

$$area\_cubo = \sum area\ faces = 6 \cdot a^2$$

$$volume\_cubo = a^3$$

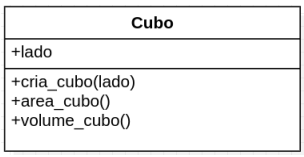

Figura 1: TAD Cubo

```c
#define _CUBO_H

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

struct cubo{
    double lado;
};

typedef struct cubo Cubo;
```

```
12
13 Cubo* cria_cubo(double);
14 double area_cubo(Cubo*);
15 double volume_cubo(Cubo*);
```

../ex12/cubo.h

```
1  #include "cubo.h"
2
3  Cubo* cria_cubo(double lado){
4      Cubo *c = (Cubo*)malloc(sizeof(Cubo));
5
6      c->lado = lado;
7      return c;
8  }
9
10 double area_cubo(Cubo *c){
11     return (6*pow(c->lado,2));
12 }
13
14 double volume_cubo(Cubo *c){
15     return pow(c->lado, 3);
16 }
```

../ex12/cubo.c

```
1  #include "cubo.h"
2
3
4  int main(int argc, char **argv){
5      double a = atof(argv[1]);
6      Cubo *c = cria_cubo(a);
7
8      printf("Area: %.4lf unidades quadradas\n", area_cubo(c));
9      printf("Volume: %.4lf unidades cubicas\n", volume_cubo(c));
10
11     return 0;
12 }
```

../ex12/main.c

Alguns testes a seguir:

```
zonzin@rodrigo:~/Documentos/Faculdade/labaedsii/labaeds2/pr3/ex12$ ./main 1
Area: 6.0000 unidades quadradas
Volume: 1.0000 unidades cubicas
zonzin@rodrigo:~/Documentos/Faculdade/labaedsii/labaeds2/pr3/ex12$ ./main 2
Area: 24.0000 unidades quadradas
Volume: 8.0000 unidades cubicas
zonzin@rodrigo:~/Documentos/Faculdade/labaedsii/labaeds2/pr3/ex12$ ./main 5
Area: 150.0000 unidades quadradas
Volume: 125.0000 unidades cubicas
zonzin@rodrigo:~/Documentos/Faculdade/labaedsii/labaeds2/pr3/ex12$ ./main 5.111
Area: 156.7339 unidades quadradas
Volume: 133.5112 unidades cubicas
```

Figura 2: Resultado

1.3

TAD para representar o conjunto de inteiros. Representado pelo tipo estruturado *struct inteiros* onde existem os atributos:

int elementos*
***int** n*
***int** ocupado*

n é um controlador de espaço para realocação de memória caso sejam inseridos mais elementos que o array *elementos* comporta.

De forma semelhante, *ocupado* é um inteiro que representa a cardinalidade do conjunto.

Detalhes matemáticos e adaptações para implementação:

- Criar conjunto vazio:
  Existência da *flag* $NaN = -9999$, pois $A = \{0, 0, \ ... \ , 0\} \neq \emptyset$.
  No nosso caso, $\emptyset := \{-9999, -9999, \ ...\}$
- União de dois conjuntos.
  Dados dois Conjuntos A e B, temos que $|A \cup B| = |A| + |B| - |A \cap B|$. No nosso caso, $|A \cup B| := max(|A| \ , \ |B|)$ por facilidade de implementação.
- Intersecção de dois conjuntos. Dado dois conjuntos A e B, $|A \cap B| = \{x : x \in A \ e \ x \in B\}$. A operação deve ser comutativa, logo $|A \cap B| = |B \cap A|$.
- Diferença de dois conjuntos.
  Operação não comutativa. Dado dois conjuntos A e B, temos que $A - B \neq B - A$. Prova: Seja $A = \{1, 2, 3, 4\}$ e $B = \{1, 3, 6, 9\}$. Temos $A - B = \{2, 4\}$ e $B - A = \{6, 9\}$. Obviamente, $A - B \neq B - A$. QED. Isso é respeitado.
- Testar se dois conjuntos são iguais.
  Se dois conjuntos A e B são iguais, então A e B têm mesma cardinalidade. Caso dois conjuntos não tenham o mesmo módulo, então podemos antecipar que são diferentes. Caso tenham a mesma cardinalidade, precisamos ainda comparar elemento a elemento dos dois conjuntos.
- Testar se um conjunto é vazio. Duas abordagens: *Lazy*: verificar se o número de elementos ocupados no conjunto é diferente de zero. Outra abordagem: verificar se os $|A|$ elementos em um conjunto A é diferente da *flag* -9999.

Figura 3: Resultado.

```c
#define _INTEIROS_H
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

struct inteiros{
    int *elemento;
    int n;
    int ocupado;
};

typedef struct inteiros Inteiros;

Inteiros* cria_conjunto_vazio(int n);
void printa_conjunto(Inteiros*);

void inserir_elemento(Inteiros*, int);
void remover_elemento(Inteiros*, int);
int pertence(Inteiros*, int);
int maior_valor(Inteiros*);
int menor_valor(Inteiros*);
int tam_inteiros(Inteiros*);

Inteiros* intersecao(Inteiros*, Inteiros*);
Inteiros* uniao(Inteiros*, Inteiros*);
int vazio(Inteiros*);
int Inteiros_iguais(Inteiros*, Inteiros*);
Inteiros* diferenca(Inteiros*, Inteiros*);

void destroi(Inteiros*);
```

../ex13/inteiros.h

```c
#include "inteiros.h"


int comparador(const void *a, const void *b){
    return (*(int*)b - *(int*)a);
}

Inteiros* cria_conjunto_vazio(int n){
    Inteiros *A = (Inteiros*)malloc(sizeof(Inteiros));
    A->elemento = (int*)malloc(sizeof(int)*n);

    for(int i = 0; i<n; i++){
        A->elemento[i] = -99999;
    }


    A->n = n;
    A->ocupado = 0;

    return A;
}

void printa_conjunto(Inteiros *A){
    printf("[");
    for(int i = 0; i < A->ocupado-1; i++){
        printf("%d,", A->elemento[i]);
    }
```

```
28      printf("%d]\n", A->elemento[A->ocupado-1]);
29  }
30
31  int menor_valor(Inteiros *A){
32      int menor = A->elemento[0];
33
34      for(int i = 0; i< A->ocupado; i++){
35          if(A->elemento[i] <= menor && A->elemento[i] != -9999) menor =
                A->elemento[i];
36      }
37
38      return menor;
39  }
40
41  int maior_valor(Inteiros *A){
42      int maior = A->elemento[0];
43
44      for(int i = 0; i< A->ocupado; i++){
45          if(A->elemento[i] >= maior) maior = A->elemento[i];
46      }
47
48      return maior;
49  }
50
51
52  void inserir_elemento(Inteiros *A, int ai){
53      if(A->ocupado == A->n){
54          A->elemento = realloc(A->elemento, sizeof(int)*A->n+10);
55          A->n = sizeof(int)*A->n+10;
56      }
57
58      for(int i = 0; i<A->ocupado; i++){
59          if(A->elemento[i] == ai) return;
60      }
61
62      A->elemento[A->ocupado] = ai;
63      A->ocupado = A->ocupado + 1;
64  }
65
66  Inteiros* uniao(Inteiros *A, Inteiros *B) {
67      Inteiros *C = cria_conjunto_vazio(A->n + B->n);
68
69      for(int i = 0; i< A->ocupado; i++){
70          inserir_elemento(C, A->elemento[i]);
71      }
72
73      for(int i = 0; i< B->ocupado; i++){
74          if(!pertence(A, B->elemento[i])){
75              inserir_elemento(C, B->elemento[i]);
76          }
77      }
78
79      return C;
80  }
81
82  Inteiros* diferenca(Inteiros *A, Inteiros *B){
83      Inteiros *dif = cria_conjunto_vazio(A->n);
84
85      for(int i = 0; i< A->ocupado; i++){
```

```
86          if(!pertence(B, A->elemento[i])){
87              inserir_elemento(dif, A->elemento[i]);
88          }
89      }
90
91      return dif;
92  }
93
94  Inteiros* intersecao(Inteiros *A, Inteiros *B){
95      Inteiros *intersec = cria_conjunto_vazio(A->n);
96
97      for (int i = 0; i < A->ocupado; i++){
98          if(pertence(B, A->elemento[i])){
99              inserir_elemento(intersec, A->elemento[i]);
100         }
101     }
102
103     return intersec;
104 }
105
106 int tam_inteiros(Inteiros*A){
107     return A->ocupado;
108 }
109
110 // RETORNA 1 SE O CONJUNTO FOR VAZIO E 0 EM CONTRARIO
111 int vazio(Inteiros *A){
112     int i = 0;
113     while(i< A->ocupado){
114         if(A->elemento[i] != -9999) return 0;
115         i++;
116     }
117
118     return 1;
119 }
120
121
122 int Inteiros_iguais(Inteiros *A, Inteiros *B){
123     if(A->ocupado != B->ocupado) return 0;
124
125     for(int i = 0; i< A->ocupado; i++){
126         if(A->elemento[i] != B->elemento[i]) return 0;
127     }
128
129     return 1;
130 }
131
132 int pertence(Inteiros *A, int ai){
133     for(int i = 0; i< A->ocupado; i++){
134         if(A->elemento[i] == ai) return 1;
135     }
136
137     return 0;
138 }
139
140 void remover_elemento(Inteiros *A, int ai){
141     for(int i = 0; i<A->ocupado; i++){
142         if(A->elemento[i] == ai){
143             A->elemento[i] = -9999;
144         }
```

```
145     }
146     //qsort(A->elemento, A->n, sizeof(int), comparador);
147 }
148
149 void destroi(Inteiros *A){
150     free(A->elemento);
151     free(A);
152 }
```

../ex13/inteiros.c

```
1  #include "inteiros.h"
2
3
4  int main(){
5
6      Inteiros *A = cria_conjunto_vazio(3);
7      inserir_elemento(A, 1);
8      inserir_elemento(A, 2);
9      inserir_elemento(A, 3);
10     inserir_elemento(A, 4);
11
12     Inteiros *B = cria_conjunto_vazio(2);
13     inserir_elemento(B, 1);
14     inserir_elemento(B, 3);
15     inserir_elemento(B, 6);
16     inserir_elemento(B, 9);
17     printf("A = ");
18     printa_conjunto(A);
19     printf("B = ");
20     printa_conjunto(B);
21     printf("\n");
22
23
24     printf("Uniao:\n");
25     Inteiros *_uniao = uniao(A, B);
26     printa_conjunto(_uniao);
27
28     printf("\nIntersecao:\n");
29     Inteiros *_inter = intersecao(A, B);
30     printa_conjunto(_inter);
31
32     printf("\nA-B:\n");
33     Inteiros *_dif1 = diferenca(A, B);
34     printa_conjunto(_dif1);
35
36     printf("\nB-A:\n");
37     Inteiros *_dif2 = diferenca(B, A);
38     printa_conjunto(_dif2);
39
40
41     printf("\nA == B? : %d\n", Inteiros_iguais(A, B));
42
43     destroi(A);
44     destroi(B);
45     destroi(_uniao);
46     destroi(_inter);
47     destroi(_dif1);
48     destroi(_dif2);
49
```

```
50
51      return 0;
52 }
```

../ex13/main.c