Atividade de Fixação:

Matheus Batista Honório

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2020

Matheus Batista Honório

Atividade de Fixação

Relatório apresentado à disciplina Estrutura de Dados do curso Engenharia de Computação do Centro de Informática, da Universidade Federal da Paraíba.

Professor: Leandro Carlos de Souza

Outubro de 2020

1 Respostas da Lista

1.1 Questão 1:

1.1.1 Resposta da a:

Estrutura de Dados é um modo particular de armazenamento e organização de dados sendo formado por um grupo de itens no qual cada item é identificado por um identificador próprio e cada um deles conhecido como um membro da estrutura. (Em várias linguagens de programação, uma estrutura é chamada "registro"e um membro é chamado "campo"). Dessa forma, objetiva-se que possam ser usados eficientemente, facilitando sua busca e modificação.

O tipo abstrato de dados (TAD) pode ser visto como um modelo matemático que encapsula um modelo de dados e um conjunto de procedimentos que atuam com exclusividade sobre os dados encapsulados. Em nível de abstração mais baixo, associado à implementação, esses procedimentos são implementados por subprogramas denominados operações, métodos ou serviços.

```
typedef struct no No;

No* conjunto_cria(int numero);
void conjunto_print(No *cabeca);
No* conjunto_uniao(No** cabeca1,No** cabeca2);
No* conjunto_cria_vazio();
int conjunto_insere(No** cabeca, int numero);
int conjunto_remove(No** cabeca, int numero);
No* conjunto_interseccao(No** cabeca1,No** cabeca2);
No* conjunto_diferenca(No** cabeca1,No** cabeca2);
int conjunto_existe_valor(No* cabeca, int numero);
int conjunto_menor_valor(No* cabeca);
int conjunto_maior_valor(No* cabeca);
int conjunto_iguais(No* cabeca1, No* cabeca2);
int conjunto_tamanho(No* cabeca);
int conjunto_eh_vazio(No* cabeca);
```

1.1.2 Resposta da b:

Array estático pode ser definido como uma estrutura homogênea de dados armazenadas contiguamente em memória. Cada variável componente é chamada de elemento e o

acesso a cada elemento do array é feito através de uma indexação da variável que vai de 0 a size_array-1.

Exemplo de Array Estático:

```
#define Eixo_X 100
#define Eixo_Y 100

int vetor[Eixo_X]; //vetor unidimensional
int matriz[Eixo_X][Eixo_Y]; //vetor matricial
```

Entretanto, o Array Dinâmico são utilizados para relacionar itens que precisam ser manipulados em tempos de execução com dimensão indefinida. Possibilitando manipular em tempo de execução funções como adicionar ou remover itens da estrutura.

Exemplo de Array Dinâmico:

```
int *p;
p = (int *) malloc(sizeof(int)); //criando o vetor dinamico
```

1.1.3 Resposta da c:

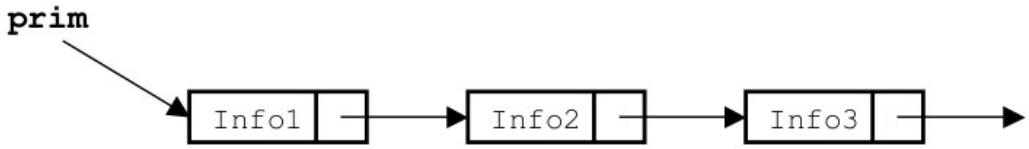
Tradicionalmente, listas são implementadas através de estruturas (associadas aos nós) armazenadas na memória dinâmica. A estrutura que implementa um nó de uma lista ligada deve incluir, além do contéudo da informação do nó, um ponteiro para o próximo nó.

Listas encadeadas, para cada novo elemento inserido na estrutura, alocamos um espaço de memória para armazená-lo. No entanto, não podemos garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo, portanto não temos acesso direto aos elementos da lista. Para que seja possível percorrer todos os elementos da lista, devemos guardar o encadeamento dos elementos, o que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista. O endereço após o último elemento aponta para NULL.

Exemplo Estrutural de uma Lista Encadeada:

```
struct no{
    int info;
    struct no* prox;
}
```

Figura 1: *Arranjo da memória de uma Lista Encadeada*

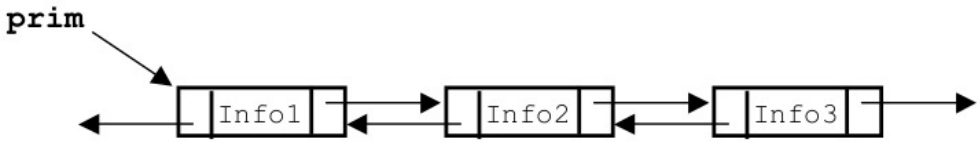


Listas duplamente encadeadas, cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior. Curiosamente, se tivermos um ponteiro para o último elemento da lista, podemos percorrer a lista em ordem inversa, bastando acessar continuamente o elemento anterior, até alcançar o primeiro elemento da lista. Tanto o anterior ao primeiro elemento quando o posterior ao ultimo apontam para NULL.

Exemplo Estrutural de uma Lista Duplamente Encadeada:

```
struct no{
    int info;
    struct no* prox;
    struct no* ante;
}
```

Figura 2: *Arranjo da memória de uma Lista Duplamente Encadeada*



Listas circulares, são estruturadas da forma que o último elemento tem como próximo o primeiro elemento da lista, formando um ciclo. A lista pode ser representada por um ponteiro para um elemento inicial qualquer da lista. Para percorrer os elementos de uma lista circular, visitamos todos os elementos a partir do ponteiro do elemento inicial até alcançarmos novamente esse mesmo elemento. Vale ressaltar que, uma lista circular pode ser tanto usada em uma lista duplamente encadeada quanto em uma lista encadeadeada simples.

Exemplo Estrutural de uma Lista Circular:

```
void conjunto_print(No *lhead, int info)
{
```

```
  No *p = lhead; /* faz p apontar para o no inicial */
  /* testa se lista nao e vazia */
  if (p)
  {
    { /* percorre os elementos ate alcancar novamente o inicio */
      do
      {
        printf("%d\n", p->info); /* imprime informacao do no */
        p = p->prox;             /* avanca para o proximo no */
      } while (p != lhead);
    }
  }
}
```

Figura 3: *Arranjo da memória de uma Lista Circular Simplesmente Encadeada*

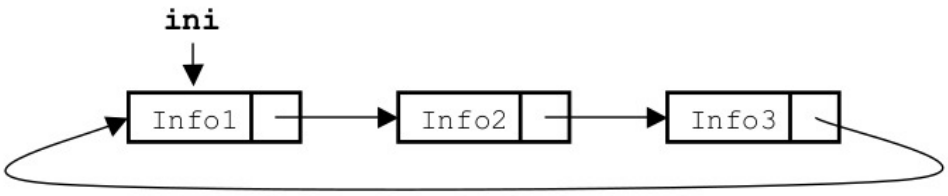
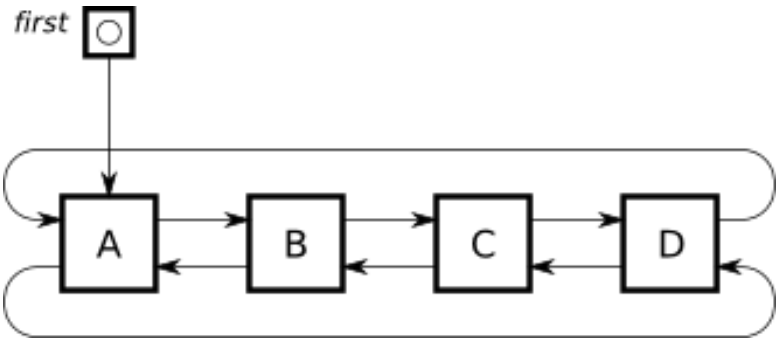


Figura 4: *Arranjo da memória de uma Lista Circular Duplamente Encadeada*



Lista heterogênea, são listas em que as informações armazenadas diferem de nó para nó. Necessita-se de três campos, sendo eles, um identificador para o objeto armazenado, um ponteiro genérico (void*) para a estrutura que contém a informação e um ponteiro para o próximo nó da lista.

Exemplo Estrutural de uma Lista Heterogênea:

```
struct No{
    int tipo;
```

```
    void* info;
    struct No *prox;
}Node;
```

1.2 Questão 2:

1.2.1 (a) União:

```
No *conjunto_uniao(No **cabeca_1, No **cabeca_2)
{
  No *aux = *cabeca_1;         //serve para percorrer o conjunto cabeca_1 sem altera-lo
  No *aux_2 = *cabeca_2;       //serve para percorrer o conjunto cabeca_2 sem altera-lo
  No *result = conjunto_cria_vazio();    //cria o conjunto resultado

  while (aux != NULL)       //enquanto aux nao for NULL percorre o loop
  {
    conjunto_insere(&result, aux->numero);    //insere o elemento no conjunto
    aux = aux->prox;                    //pula para o proximo termo do conjunto
  }

  while (aux_2 != NULL)     //enquanto aux_2 nao for NULL percorre o loop
  {
    conjunto_insere(&result, aux_2->numero);    //insere o elemento no conjunto
    aux_2 = aux_2->prox;    //pula para o proximo termo do conjunto
  }

  return result;        //retorna o conjunto uniao
}
```

1.2.2 (b) Cria um conjunto vazio:

```
No *conjunto_cria_vazio()
{
  No *no = (No *)malloc(sizeof(No)); /*aloca memoria necessaria para o noh e
  converte para No*.*/
  if (no)
  {
```

```
    no = NULL;    //noh igual a NULL
  }
  return no; //retorna o noh vazio criado.
}
```

1.2.3 (c) Insere:

```
int conjunto_insere(No **cabeca, int numero)
{
  No *aux = conjunto_cria(numero);        /*cria um conjunto auxiliar com o numero
  do parametro como elemento*/
  if (conjunto_existe_valor(*cabeca, numero))     /*se ja existir o valor do parametro
   no conjunto*/
  {
    return 0;       /*retorna 0 porque nao precisa ser setado, pois
    ja tem no conjunto*/
  }
  if (aux)    //se aux tiver sido criado sem problemas
  {
    aux->prox = *cabeca;    /*o proximo elemento de auxiliar vai ser a cabeca da lista,
    fazendo com o que aux seja o primeiro elemento do conjunto*/
    *cabeca = aux;    //cabeca do conjunto igual aux
    return 1;
  }
}
```

1.2.4 (d) Remove:

```
int conjunto_remove(No **cabeca, int numero)
{
  //cria os noh auxiliares e um validador
  No *ante = NULL;
  No *aux = *cabeca;
  int found = 0;    //encontrado igual a 0
  while (aux != NULL)   //percorre o conjunto enquanto aux for diferente de NULO
  {
    if (aux->numero == numero)    //se o numero de aux for igual ao numero do parametro
```

```
10      {
11        found = 1;    //encontrado igual a verdadeiro e sai do loop
12        break;
13      }
14      ante = aux;   //anterior igual ao elemento atual
15      aux = aux->prox;    //aux igual ao proximo elemento
16    }
17
18    if (found)    //Se o numero for encontrado
19    {
20      if (ante == NULL)   //se anterior for NULL
21      {
22        *cabeca = aux->prox;    //seta cabeca como o proximo elemento de aux
23        free(aux);    //libera aux
24        return 1;   //retorna 1
25      }
26
27      ante->prox = aux->prox;   /*proximo elemento do anterior eh igual ao proximo
28      elemento do aux*/
29      free(aux);    //libera aux
30      return 1;   //retorna verdadeiro
31    }
32
33    return 0;     //retorna falso
34  }
```

1.2.5 (e) Intersecção:

```
1   No *conjunto_interseccao(No **cabeca1, No **cabeca2)
2   {
3     //cria o conjunto que ira retornar e o aux para percorrer o conjunto
4     No *resultante = conjunto_cria_vazio();
5     No *aux = *cabeca1;
6     No *aux_2 = *cabeca2;
7
8     if (conjunto_tamanho(*cabeca1) <= conjunto_tamanho(*cabeca2))   /*Se o tamanho do
9     primeiro conjunto menor que o do segundo*/
10    {
11      while (aux != NULL)       //enquanto aux diferente de NULL
```

```
12      {
13        if (conjunto_existe_valor(*cabeca2, aux->numero))   /*Se o numero de aux existir
14        na cabeca2*/
15        {
16          conjunto_insere(&resultante, aux->numero);    //insere o numero no conjunto
17        }
18
19        aux = aux->prox;    //pula para o proximo elemento do conjunto
20      }
21      return resultante;    //retorna o conjunto resultante
22    }
23    else
24    {
25      while (aux_2 != NULL)     //enquanto aux diferente de NULL
26      {
27        if (conjunto_existe_valor(*cabeca1, aux_2->numero))     /*se o numero de aux2
28        existir na cabeca1*/
29        {
30          conjunto_insere(&resultante, aux_2->numero);      /*insere o numero de aux_2
31          no resultante*/
32        }
33
34        aux_2 = aux_2->prox;    //pula para o proximo elemento do conjunto
35      }
36      return resultante;
37    }
38  }
```

1.2.6 (f) Diferença:

```
1   No *conjunto_diferenca(No **cabeca1, No **cabeca2)
2   {
3     No *res_diferenca = conjunto_cria_vazio();  //cria o conjunto resultado
4     No *aux = *cabeca1;       //serve para percorrer o conjunto cabeca1 sem altera-lo
5     No *aux_2 = *cabeca2;     //serve para percorrer o conjunto cabeca2 sem altera-lo
6
7     while (aux != NULL)       //enquanto aux nao for NULL percorre o loop
8     {
9       if (!conjunto_existe_valor(aux_2, aux->numero))   /*Se nao existir o valor passado
```

```
    em aux_2*/
    {
      conjunto_insere(&res_diferenca, aux->numero);   //insere no conjunto diferenca
    }
    aux = aux->prox;        //pula para o proximo termo do conjunto
  }
  return res_diferenca;     //retorna o conjunto diferenca
}
```

1.2.7 (g) Testa se um número pertence ao conjunto:

```
int conjunto_existe_valor(No *cabeca, int numero)
{
  if (conjunto_eh_vazio(cabeca)) //se cabeca diferente de NULO
  {
    return 0;
  }
  while (cabeca != NULL) //enquanto cabeca for diferente de NULL
  {
    if (cabeca->numero == numero) /*se o numero do elemento for igual ao numero setado
    como parametro.*/
    {
      return 1; //retorna verdadeiro
    }
    cabeca = cabeca->prox; //avanca para o proximo elemento da lista
  }
  return 0;
}
```

1.2.8 (h) Menor valor:

```
int conjunto_menor_valor(No *cabeca)
{
  int menor = cabeca->numero; //menor comeca com o numero do primeiro elemento
  while (cabeca != NULL)      //enquanto cabeca do conjunto diferente de NULL.
  {
    if (cabeca->numero < menor)   /*se o numero da cabeca for menor que o menor numero
```

```
    encontrado ate o momento*/
    {
      menor = cabeca->numero; /*Se o elemento numero atual da tabela for menor que o
      menor setado anteriormente, atribui o numero atual como menor*/
    }
    cabeca = cabeca->prox; //cabeca igual ao proximo elemento do conjunto.
  }

  return menor; //retorna o menor.
}
```

1.2.9 (i) Maior valor:

```
int conjunto_maior_valor(No *cabeca)
{
  int maior = cabeca->numero; //maior comeca com o numero do primeiro elemento
  while (cabeca != NULL)      //enquanto cabeca do conjunto diferente de NULL.
  {
    if (cabeca->numero > maior)     /*se o numero da cabeca for maior que o maior numero
    encontrado ate o momento*/
    {
      maior = cabeca->numero; /*Se o elemento numero atual da tabela for maior que o
      maior setado anteriormente, atribui o numero atual como maior*/
    }
    cabeca = cabeca->prox; //cabeca igual ao proximo elemento do conjunto.
  }

  return maior; //retorna o maior.
}
```

1.2.10 (j) Testa se os conjuntos são iguais:

```
int conjunto_iguais(No *cabeca1, No *cabeca2)
{
  if (cabeca1) //cabeca diferente de 0;
  {
    while (cabeca1 != NULL)
```

```
    {
      if (cabeca1->numero != cabeca2->numero)
      {           //se o elemento do conjunto 1 diferente do elemento do conjunto 2
        return 0; //sao diferentes e retorna 0 = falso
      }
      cabeca1 = cabeca1->prox; //aponta o proximo para percorrer o conjunto 1
      cabeca2 = cabeca2->prox; //aponta o proximo para percorrer o conjunto 2
    }
    return 1;
  }
  else    //falso para se o conjunto nao existir ou estiver NULL
  {
    printf("conjunto 1 vazio.\n");
    return 0;
  }
}
```

1.2.11 (k) Retorna o Tamanho do conjunto:

```
int conjunto_tamanho(No *cabeca)
{
  int contador = 0;       //contador do tamanho do conjunto.
  while (cabeca != NULL) //enquanto cabeca do conjunto diferente de NULL.
  {
    contador++;             //contador +1 a cada verificacao.
    cabeca = cabeca->prox; //cabeca igual ao proximo elemento do conjunto.
  }

  return contador; //retorna resultado do contador
}
```

1.2.12 (l) Testa se o conjunto é vazio:

```
int conjunto_eh_vazio(No *cabeca)
{
  return (cabeca == NULL); /*se o No* tiver os elementos 0 e NULO,
  retorna 1 se verdadeiro e 0 para falso.*/
```

```
}
```

1.2.13 (m) Faça um programa de teste para o seu TAD:

```
#include <stdio.h>
#include <stdlib.h>
#include "Q2_Struct.h"


int main (){

  //conjunto X//
  No* x = conjunto_cria_vazio();
  conjunto_insere(&x, 4);
  conjunto_insere(&x, (-7));
  conjunto_insere(&x, 20);


  //conjunto Y//
  No* y = conjunto_cria(100);
  conjunto_insere(&y, -20);
  conjunto_insere(&y, 20);
  conjunto_insere(&y, 20);
  conjunto_insere(&y, 20);


  //conjunto Z//
  No* z = conjunto_cria(-20);
  conjunto_insere(&z, 15000);
  conjunto_insere(&z, 0);
  conjunto_insere(&z, 20);
  conjunto_insere(&z, 100);
  conjunto_insere(&z, 152);
  conjunto_insere(&z, 70);


  printf("-------Printando os conjuntos-------\n");
  printf("-----Lista X-----\n");
  conjunto_print(x);
```

```
36
37    printf("-----Lista Y-----\n");
38    conjunto_print(y);
39
40    printf("-----Lista Z-----\n");
41    conjunto_print(z);
42
43
44    //Tamanho dos Conjuntos//
45    int tamanho;
46    printf("\n-----Printando os Tamanhos dos Conjuntos-----\n");
47    tamanho = conjunto_tamanho(x);
48    printf("Tamanho do conjunto: %d \n", tamanho);
49
50    tamanho = conjunto_tamanho(y);
51    printf("Tamanho do conjunto: %d \n", tamanho);
52
53    tamanho = conjunto_tamanho(z);
54    printf("Tamanho do conjunto: %d \n", tamanho);
55
56
57    //Teste qual o maior numero//
58    printf("\n-----Printando o Maior Numero dos Conjuntos-----\n");
59    printf("maior numero do conjunto x: %d.\n", conjunto_maior_valor(x));
60    printf("maior numero do conjunto y: %d.\n", conjunto_maior_valor(y));
61    printf("maior numero do conjunto z: %d.\n", conjunto_maior_valor(z));
62
63
64    //Teste qual o menor numero//
65    printf("\n-----Printando o Menor Numero dos Conjuntos-----\n");
66    printf("menor numero do conjunto x: %d.\n", conjunto_menor_valor(x));
67    printf("menor numero do conjunto y: %d.\n", conjunto_menor_valor(y));
68    printf("menor numero do conjunto z: %d.\n", conjunto_menor_valor(z));
69
70
71    //Teste se existe o numero//
72    printf("\n-----Printando se Existe os Valores nos Conjuntos-----\n");
73    if(conjunto_existe_valor(x, 15)){
74      printf("existe.\n");
75    }else{
```

```
76      printf("nao existe.\n");
77    }
78
79    if(conjunto_existe_valor(y, 0)){
80      printf("existe.\n");
81    }else{
82      printf("nao existe.\n");
83    }
84
85    if(conjunto_existe_valor(z, 100)){
86      printf("existe.\n");
87    }else{
88      printf("nao existe.\n");
89    }
90
91
92    //Teste se os conjuntos sao iguais//
93    printf("\n-----Printando se os Conjuntos sao Iguais-----\n");
94    if (conjunto_iguais(x, y)){
95      printf("conjuntos iguais.\n");
96    }else{
97      printf("conjuntos diferentes.\n");
98    }
99
100   if (conjunto_iguais(z, y)){
101     printf("conjuntos iguais.\n");
102   }else{
103     printf("conjuntos diferentes.\n");
104   }
105
106   if (conjunto_iguais(y, y)){
107     printf("conjuntos iguais.\n");
108   }else{
109     printf("conjuntos diferentes.\n");
110   }
111
112
113   //Criando os conjuntos de teste para serem atribuidos os conjuntos abaixo//
114   No* t1 = conjunto_cria_vazio();
115   No *t2 = conjunto_cria_vazio();
```

```
116     No *t3 = conjunto_cria_vazio();
117     No *t4 = conjunto_cria_vazio();
118     No *t5 = conjunto_cria_vazio();
119     No *t6 = conjunto_cria_vazio();
120     No *t7 = conjunto_cria_vazio();
121     No *t8 = conjunto_cria_vazio();
122     No *t9 = conjunto_cria_vazio();
123
124
125     //Teste se o conjunto eh vazio//
126     printf("\n-----Printando se tem Conteudo no Conjunto-----\n");
127     if (conjunto_eh_vazio(t1)){
128       printf("conjunto vazio.\n");
129     }else{
130       printf("conjunto cheio.\n");
131     }
132
133     if (conjunto_eh_vazio(x)){
134       printf("conjunto vazio.\n");
135     }else{
136       printf("conjunto cheio.\n");
137     }
138
139
140     //Retorna o conjunto de Interseccao//
141     printf("\n-----Printando a Interseccao dos Conjuntos-----\n");
142     printf("\n--elementos (y, z):\n");
143     t1 = conjunto_interseccao(&y, &z);
144     conjunto_print(t1);
145
146     printf("\n--elementos (z, x):\n");
147     t2 = conjunto_interseccao(&z, &x);
148     conjunto_print(t2);
149
150     printf("\n--elementos (x, y):\n");
151     t3 = conjunto_interseccao(&x, &y);
152     conjunto_print(t3);
153
154
155     //Retorna o conjunto de Diferenca//
```

```
156     printf("\n-----Printando a Diferenca dos Conjuntos-----\n");
157     printf("\n--elementos (z, y):\n");
158     t4 = conjunto_diferenca(&z, &y);
159     conjunto_print(t4);
160 
161     printf("\n--elementos (x, y):\n");
162     t5 = conjunto_diferenca(&x, &y);
163     conjunto_print(t5);
164 
165     printf("\n--elementos (x, z):\n");
166     t6 = conjunto_diferenca(&x, &z);
167     conjunto_print(t6);
168 
169 
170     //Retorna a Uniao dos Conjuntos//
171     printf("\n-----Printando a Uniao dos Conjuntos-----\n");
172     printf("\n--elementos (z, y):\n");
173     t7 = conjunto_uniao(&z, &y);
174     conjunto_print(t7);
175 
176     printf("\n--elementos (x, y):\n");
177     t8 = conjunto_uniao(&y, &x);
178     conjunto_print(t8);
179 
180     printf("\n--elementos (z, x):\n");
181     t9 = conjunto_uniao(&z, &x);
182     conjunto_print(t9);
183 
184 
185     //Remove o elemento do conjunto//
186     conjunto_remove(&x, -7);
187     conjunto_remove(&y, 20);
188     conjunto_remove(&z, 0);
189     conjunto_remove(&z, 15000);
190     conjunto_remove(&z, 152);
191 
192     printf("\n-----Printando os Conjuntos depois do Remover-----\n");
193     printf("\n-----Lista X-----\n");
194     conjunto_print(x);
195 
```

```
196   printf("\n-----Lista Y-----\n");
197   conjunto_print(y);
198
199   printf("\n-----Lista Z-----\n");
200   conjunto_print(z);
201
202
203   //Libera os Conjuntos Criados//
204   free(x); free(y); free(z);
205   free(t1); free(t2); free(t3);
206   free(t4); free(t5); free(t6);
207   free(t7); free(t8); free(t9);
208
209   return 0;
210 }
```

1.3 Questao 3:

Definindo os structs da Questão:

```
1  typedef struct NO
2  {
3    float valor;
4    int coluna;
5    struct NO *prox;
6  } NO;
7
8  typedef NO *PONT;
9
10 typedef struct MATRIZ
11 {
12   PONT *p_line;
13   int linhas;
14   int colunas;
15 } MATRIZ;
```

1.3.1 (a) Criar a matriz esparsa:

```
MATRIZ *cria_matriz(int linhas, int colunas)
{
  // Aloca espaco para a matriz e o ponteiro de linhas
  MATRIZ *matriz = (MATRIZ *)malloc(sizeof(MATRIZ));
  matriz->p_line = (PONT *)malloc(linhas * sizeof(PONT));

  // Se matriz existir
  if (matriz)
  {
    // Atribui os parametros a matriz criada
    matriz->linhas = linhas;
    matriz->colunas = colunas;

    // Percorre todas as linhas para inicializa-las com valor NULL
    for (int i = 0; i < linhas; i++)
    {
      matriz->p_line[i] = NULL;
    }
  }
  return matriz;
}
```

1.3.2 (b) Remover a matriz esparsa:

```
void remover_matriz(MATRIZ *matriz)
{
  free(matriz); // Libera a matriz
}
```

1.3.3 (c) Inserir um valor na posição(i,j):

```
int inserir_no(MATRIZ *matriz, int linha, int coluna, float valor)
{
  // Verifica se a coordenada da matriz eh valida, se for, retorna false
```

```
4   if (linha - 1 < 0 || linha - 1 >= matriz->linhas || coluna - 1 < 0 || coluna - 1 >= ma
5   {
6     return 0;
7   }
8
9   // Seta o ponteiro anterior como NULL
10  PONT ant = NULL;
11  // Seleciona a linha correspondente a enviada como parametro
12  PONT atual = matriz->p_line[linha - 1];
13
14  /* Percorre os elementos ate o valor da coluna correspondente ser igual ao do
15  parametro enviado*/
16  while (atual != NULL && atual->coluna < coluna - 1)
17  {
18    ant = atual;
19    atual = atual->prox;
20  }
21
22  // Entra nesse if se o elemento onde quer add for diferente de NULL
23  if (atual != NULL && atual->coluna == coluna - 1)
24  {
25    // Tratamento para valor igual a 0 que nao precisa ser alocado na memoria
26    if (valor == 0)
27    {
28      if (ant == NULL)
29        matriz->p_line[linha - 1] = atual->prox;
30      else
31        ant->prox = atual->prox;
32
33      free(atual); // Libera o espaco do elemento
34    }
35    else{
36      // Se o valor nao for 0, faz a atribuicao
37      atual->valor = valor;
38    }
39  }
40
41  else if (valor != 0)
42  {
43    // Armazena espaco pro novo elemento
```

```
    PONT novo = (PONT)malloc(sizeof(NO));

    // Atribui os valores
    novo->coluna = coluna - 1;
    novo->valor = valor;
    novo->prox = atual;

    // Adiciona o elemento dependendo do valor do anterior a posicao desejada
    if (ant == NULL)
      matriz->p_line[linha - 1] = novo;

    else
      ant->prox = novo;
  }
  return 1; // Retorna verdadeiro se a operacao for um sucesso
}
```

1.3.4 (d) Retornar um valor na posição (i,j):

```
float acessar_no(MATRIZ *matriz, int linha, int coluna)
{
  /* Se os valores passados nao corresponderem a matriz, ou seja, uma coordenada
  invalida, retorna 0 */
  if (linha - 1 < 0 || linha - 1 >= matriz->linhas || coluna - 1 < 0 || coluna - 1 >= ma
  {
    return 0;
  }
  // Seleciona a linha correspondente a enviada como parametro
  PONT atual = matriz->p_line[linha - 1];

  // Percorre os elementos ate a coluna ser igual ao do paramento enviado
  while (atual != NULL && atual->coluna < coluna - 1){
    atual = atual->prox;
  }

  // Verifica se eh o elemento que quer acessar
  if (atual != NULL && atual->coluna == coluna - 1){
    // Retorna o valor do elemento
    return atual->valor;
```

```
  }
  return 0;
}
```

1.3.5 (d) Remover um valor na posição (i,j):

```
int remover_no(MATRIZ *matriz, int linha, int coluna)
{
  // Verifica se a coordenada da matriz eh valida, se for, retorna false
  if (linha - 1 < 0 || linha - 1 >= matriz->linhas || coluna - 1 < 0 || coluna - 1 >= ma
  {
    return 0;
  }
  // Seleciona a linha correspondente a enviada como parametro
  PONT atual = matriz->p_line[linha - 1];

  // Percorre os elementos ate a coluna ser igual ao do paramento enviado
  while (atual != NULL && atual->coluna < coluna - 1){
    atual = atual->prox;
  }

  // Verifica se eh o elemento que quer excluir
  if (atual != NULL && atual->coluna == coluna - 1)
  {
    free(atual); // Libera o espaco de memoria do mesmo
    return 1;
  }

  return 0;
}
```

1.3.6 (e) Exibir na tela a matriz (com todos os seus 0s não armazenados):

```
void print_matriz(MATRIZ *matriz)
{
  //Pega o total de linhas e colunas
  int total_linhas = matriz->linhas;
```

```c
  int total_colunas = matriz->colunas;

  // Percorre todas as linhas
  for (int i = 0; i < total_linhas; i++)
  {
    // Percorre todas as colunas da linha correspondente
    for (int j = 0; j < total_colunas; j++)
    {
      // Printa o valor da posicao
      printf("%.1f  ", acessar_no(matriz, i + 1, j + 1));
    }
    printf("\n");
  }
}
```

1.3.7 (f) Faça um programa de teste para seu TAD:

```c
#include <stdio.h>
#include <stdlib.h>
#include "Q3.h"

int main (){

  /*Linhas e Colunas foram adaptadas com -1
  para nao serem um indice a mais */

  //Criacao da Matriz
  MATRIZ* m1 = cria_matriz(15, 5);
  MATRIZ* m2 = cria_matriz(45, 15);


  printf("------Atribuindo Elementos------\n");
  if(inserir_no(m1, 10, 3, 72.5)){
    printf("Adicionado com sucesso\n");
  }else{
    printf("Falha ao adicionar, coordenada da matriz invalida\n");
  }

  if(inserir_no(m1, 16, 3, 195.59)){
```

```
23      printf("Adicionado com sucesso\n");
24    }else{
25      printf("Falha ao adicionar, coordenada da matriz invalida\n");
26    }
27
28    if(inserir_no(m1, 2, 3, 3.14156)){
29      printf("Adicionado com sucesso\n");
30    }else{
31      printf("Falha ao adicionar, coordenada da matriz invalida\n");
32    }
33
34    if(inserir_no(m1, 9, 3, 72.5)){
35      printf("Adicionado com sucesso\n");
36    }else{
37      printf("Falha ao adicionar, coordenada da matriz invalida\n");
38    }
39
40    if(inserir_no(m1, 8, 2, 56149.554442)){
41      printf("Adicionado com sucesso\n");
42    }else{
43      printf("Falha ao adicionar, coordenada da matriz invalida\n");
44    }
45
46    if(inserir_no(m1, 2, 8, 9.16849)){
47      printf("Adicionado com sucesso\n");
48    }else{
49      printf("Falha ao adicionar, coordenada da matriz invalida\n");
50    }
51
52    printf("\n------Printando os elementos da Matriz------\n");
53    printf("Elemento da posicao: %.2f\n", acessar_no(m1,9,10));
54    printf("Elemento da posicao: %.2f\n", acessar_no(m1,2,3));
55    printf("Elemento da posicao: %.2f\n", acessar_no(m1,2,8));
56    printf("Elemento da posicao: %.2f\n", acessar_no(m1,16,3));
57    printf("Elemento da posicao: %.2f\n", acessar_no(m1,8,2));
58
59    printf("\n------Printando todos os elementos da Matriz------\n");
60    print_matriz(m1);
61
62    printf("\n------Removendo elementos da Matriz------\n");
```

```
63    if(remover_no(m1, 2, 8)){
64      printf("Elemento removido com Sucesso\n");
65    }else{
66      printf("Erro na remocao, coordenada invalida\n");
67    }
68
69    if(remover_no(m1, 2, 3)){
70      printf("Elemento removido com Sucesso\n");
71    }else{
72      printf("Erro na remocao, coordenada invalida\n");
73    }
74
75
76    printf("\n------Printando todos os elementos da Matriz------\n");
77    print_matriz(m1);
78
79
80    printf("\n------Fim do programa e removendo a Matriz da Memoria------\n");
81    remover_matriz(m1);
82    remover_matriz(m2);
83
84
85    return 0;
86  }
```