# Lista 01 de exercícios AED

Marcelo Corni Alves

19 de março de 2024

Códigos das implementações da lista de exercícios disponíveis em
`https://github.com/marcelocorniufjf/Lista01Code`

## Listas, Pilhas e Filas

**1-)** Para cada estrutura abaixo, implemente os métodos contido(K,L), inserir(K,L) e remover(K,L) e verifique as complexidades de cada método.

No caso de filas e pilhas, o remover(K,L) não terá o argumento K, visto que filas e pilhas removem sempre quem está na extremidade.

**a-)** Lista duplamente encadeada: considere a inserção sempre no final da lista

```
1  public class No
2  {
3    public int chave;
4    public No proximo;
5    public No anterior;
6
7    public No(int chave)
8    {
9      this.chave = chave;
10     proximo = null;
11     anterior = null;
12   }
13 }
14
15 public class ListaDuplamenteEncadeada
16 {
17   private No cabeca;
18   private No cauda;
19
20   public ListaDuplamenteEncadeada()
```

```
21    {
22      cabeca = null;
23      cauda = null;
24    }
25
26    public bool Contido(int chave)
27    {
28      No atual = cabeca;
29      while (atual != null)
30      {
31        if (atual.chave == chave)
32        {
33          return true;
34        }
35        atual = atual.proximo;
36      }
37      return false;
38    }
39
40    public void Inserir(int chave)
41    {
42      No novoNo = new No(chave);
43      if (cabeca == null)
44      {
45        cabeca = novoNo;
46        cauda = novoNo;
47      }
48      else
49      {
50        cauda.proximo = novoNo;
51        novoNo.anterior = cauda;
52        cauda = novoNo;
53      }
54    }
55
56    public void Remover(int chave)
57    {
58      No atual = cabeca;
59      while (atual != null)
60      {
61        if (atual.chave == chave)
62        {
63          if (atual.anterior != null)
64          {
65            atual.anterior.proximo = atual.proximo;
66          }
67          else
68          {
69            cabeca = atual.proximo;
```

```
70          }
71          if (atual.proximo != null)
72          {
73            atual.proximo.anterior = atual.anterior;
74          }
75          else
76          {
77            cauda = atual.anterior;
78          }
79          return;
80        }
81        atual = atual.proximo;
82      }
83    }
84  }
```

**Complexidades:** A complexidade de tempo para Contido(K,L) é O(n), para Inserir(K,L) é O(1), e para Remover(L) é O(n), onde n é o número de elementos na lista.

**b-)** Fila com lista simplesmente encadeada: considere que vc tem uma variável head e uma tail, onde head marca a cabeça e tail o ultimo nó na cauda.

```
1   public class No
2   {
3     public int chave;
4     public No proximo;
5
6     public No(int chave)
7     {
8       this.chave = chave;
9       this.proximo = null;
10    }
11  }
12
13  public class Fila
14  {
15    private No cabeca;
16    private No cauda;
17
18    public Fila()
19    {
20      this.cabeca = null;
21      this.cauda = null;
22    }
```

```
23
24    public bool Contido(int chave)
25    {
26      No atual = this.cabeca;
27      while (atual != null)
28      {
29        if (atual.chave == chave)
30        {
31          return true;
32        }
33        atual = atual.proximo;
34      }
35      return false;
36    }
37
38    public void Inserir(int chave)
39    {
40      No novoNo = new No(chave);
41      if (this.cabeca == null)
42      {
43        this.cabeca = novoNo;
44        this.cauda = novoNo;
45      }
46      else
47      {
48        this.cauda.proximo = novoNo;
49        this.cauda = novoNo;
50      }
51    }
52
53    public void Remover()
54    {
55      if (this.cabeca != null)
56      {
57        this.cabeca = this.cabeca.proximo;
58        if (this.cabeca == null)
59        {
60          this.cauda = null;
61        }
62      }
63    }
64  }
```

**Complexidades:** A complexidade de tempo para Contido(K,L) é O(n), para Inserir(K,L) é O(1), e para Remover(L) é O(1), onde n é o número de elementos na fila.

**c-)** Pilha com lista simplesmente encadeada

```
1  public class No
2  {
3    public int chave;
4    public No proximo;
5
6    public No(int chave)
7    {
8      this.chave = chave;
9      this.proximo = null;
10   }
11 }
12
13 public class Pilha
14 {
15   private No topo;
16
17   public Pilha()
18   {
19     this.topo = null;
20   }
21
22   public bool Contido(int chave)
23   {
24     No atual = this.topo;
25     while (atual != null)
26     {
27       if (atual.chave == chave)
28       {
29         return true;
30       }
31       atual = atual.proximo;
32     }
33     return false;
34   }
35
36   public void Inserir(int chave)
37   {
38     No novoNo = new No(chave);
39     if (this.topo == null)
40     {
41       this.topo = novoNo;
42     }
43     else
44     {
45       novoNo.proximo = this.topo;
46       this.topo = novoNo;
47     }
```

```
48    }
49
50    public void Remover()
51    {
52      if (this.topo != null)
53      {
54        this.topo = this.topo.proximo;
55      }
56    }
57  }
```

**Complexidades:** A complexidade de tempo para Contido(K,L) é O(n), para Inserir(K,L) é O(1), e para Remover(L) é O(1), onde n é o número de elementos na pilha.

**d-)** Lista duplamente encadeada circular

```
1   public class No
2   {
3     public int chave;
4     public No proximo;
5     public No anterior;
6
7     public No(int chave)
8     {
9       this.chave = chave;
10      this.proximo = null;
11      this.anterior = null;
12    }
13  }
14
15  public class ListaDuplamenteEncadeadaCircular
16  {
17    private No cabeca;
18    private No cauda;
19
20    public ListaDuplamenteEncadeadaCircular()
21    {
22      this.cabeca = null;
23      this.cauda = null;
24    }
25
26    public bool Contido(int chave)
27    {
28      No atual = this.cabeca;
29
30
```

```
31      while (atual != null && atual != this.cauda)
32      {
33        if (atual.chave == chave)
34        {
35          return true;
36        }
37        atual = atual.proximo;
38      }
39      if (this.cauda != null && this.cauda.chave == chave)
40      {
41        return true;
42      }
43      return false;
44    }
45
46    public void Inserir(int chave)
47    {
48      No novoNo = new No(chave);
49      if (this.cabeca == null)
50      {
51        this.cabeca = novoNo;
52        this.cauda = novoNo;
53        novoNo.proximo = novoNo;
54        novoNo.anterior = novoNo;
55      }
56      else
57      {
58        novoNo.proximo = this.cabeca;
59        novoNo.anterior = this.cauda;
60        this.cabeca.anterior = novoNo;
61        this.cauda.proximo = novoNo;
62        this.cauda = novoNo;
63      }
64    }
65
66    public void Remover(int chave)
67    {
68      No atual = this.cabeca;
69      while (atual != null && atual != this.cauda)
70      {
71        if (atual.chave == chave)
72        {
73          atual.anterior.proximo = atual.proximo;
74          atual.proximo.anterior = atual.anterior;
75          if (atual == this.cabeca)
76          {
77            this.cabeca = atual.proximo;
78          }
79
```

```
80            if (atual == this.cauda)
81            {
82              this.cauda = atual.anterior;
83            }
84           return;
85         }
86         atual = atual.proximo;
87       }
88       if (this.cauda != null && this.cauda.chave == chave)
89       {
90         this.cauda.anterior.proximo = this.cabeca;
91         this.cabeca.anterior = this.cauda.anterior;
92         this.cauda = this.cauda.anterior;
93       }
94     }
95   }
```

**Complexidades:** A complexidade de tempo para Contido(K,L) é O(n), para Inserir(K,L) é O(1), e para Remover(L) é O(n) no pior caso (quando o elemento a ser removido está no final da lista).

**2-)** Listas são usadas para representar números muito grandes (p.ex, com 1000 dígitos), uma vez que seria impossível representá-lo em máquinas de 64bits.

Para representar inteiros grandes com listas, é usada uma representação em que cada dígito do inteiro é armazenado em um nó da lista.

Considere duas listas encadeadas L1 e L2 representando números grandes (cada digito por nó).

Faça um algoritmo que faça a soma de dois inteiros grandes e retorne a lista L3 = L1 + L2.

```
1  public class No
2  {
3    public int digito;
4    public No proximo;
5
6    public No(int digito)
7    {
8      this.digito = digito;
9      this.proximo = null;
10   }
11 }
12
13 public class ListaInteiroGrande
```

```
14 {
15   private No cabeca;
16
17   public ListaInteiroGrande()
18   {
19     this.cabeca = null;
20   }
21
22   public void InserirDigito(int digito)
23   {
24     No novoNo = new No(digito);
25     if (this.cabeca == null)
26     {
27       this.cabeca = novoNo;
28     }
29     else
30     {
31       novoNo.proximo = this.cabeca;
32       this.cabeca = novoNo;
33     }
34   }
35
36   public ListaInteiroGrande Soma(ListaInteiroGrande lista1,
37                                  ListaInteiroGrande lista2)
38   {
39     ListaInteiroGrande resultado = new ListaInteiroGrande();
40     No atual1 = lista1.cabeca;
41     No atual2 = lista2.cabeca;
42     int somatorio = 0;
43
44     while (atual1 != null || atual2 != null || somatorio != 0)
45     {
46       int soma = somatorio;
47       if (atual1 != null)
48       {
49         soma += atual1.digito;
50         atual1 = atual1.proximo;
51       }
52       if (atual2 != null)
53       {
54         soma += atual2.digito;
55         atual2 = atual2.proximo;
56       }
57       somatorio = soma / 10;
58       int digitoSoma = soma % 10;
59       resultado.InserirDigito(digitoSoma);
60     }
61     return resultado;
62   }
```

```
63
64    public void Imprimir()
65    {
66      No atual = this.cabeca;
67      while (atual != null)
68      {
69        Console.Write(atual.digito);
70        atual = atual.proximo;
71      }
72      Console.WriteLine();
73    }
74  }
```

**3-)** Seja L uma lista simplesmente encadeada. Escreva um algoritmo que, percorrendo a lista uma única vez, constrói:

A. Uma lista L' que possui os valores de L em ordem inversa

B. Uma lista L" que possui a metade dos nós da lista L, onde o primeiro nó de L" contém a soma do primeiro nó de L com o último nó de L, o segundo nó de L' contém a soma do segundo nó de L com o penúltimo nó de L e assim por diante: L"= < L1+Ln, L2+Ln-1, L3+Ln-2, ... , Ln/2 + Ln/2+1>, onde n é sempre par.

```
1   public class No
2   {
3     public int Valor { get; set; }
4     public No Proximo { get; set; }
5
6     public No(int valor)
7     {
8       Valor = valor;
9       Proximo = null;
10    }
11  }
12
13  public class ListaEncadeada
14  {
15    private No cabeca;
16    private int contador;
17
18    public ListaEncadeada()
19    {
20      cabeca = null;
21      contador = 0;
22    }
23
```

```
24    public void Inserir(int valor)
25    {
26      No novoNo = new No(valor);
27      if (cabeca == null)
28      {
29        cabeca = novoNo;
30      }
31      else
32      {
33        No atual = cabeca;
34        while (atual.Proximo != null)
35        {
36          atual = atual.Proximo;
37        }
38        atual.Proximo = novoNo;
39      }
40      contador++;
41    }
42
43    public void Imprimir()
44    {
45      No atual = cabeca;
46      while (atual != null)
47      {
48        Console.Write(atual.Valor + "␣");
49        atual = atual.Proximo;
50      }
51      Console.WriteLine();
52    }
53
54    public ListaEncadeada InverterLista()
55    {
56      ListaEncadeada listaReversa = new ListaEncadeada();
57      No atual = cabeca;
58      while (atual != null)
59      {
60        listaReversa.InserirNaFrente(atual.Valor);
61        atual = atual.Proximo;
62      }
63      return listaReversa;
64    }
65
66    public ListaEncadeada SomarMeiaLista()
67    {
68      ListaEncadeada somarMeiaLista = new ListaEncadeada();
69      No atual = cabeca;
70      for (int i = 0; i < contador / 2; i++)
71      {
72
```

```
73          int sum = atual.Valor
74                    + PegarNoNaPosicao(contador - i - 1).Valor;
75          somarMeiaLista.Inserir(sum);
76          atual = atual.Proximo;
77        }
78        return somarMeiaLista;
79      }
80
81      private void InserirNaFrente(int valor)
82      {
83        No novoNo = new No(valor);
84        novoNo.Proximo = cabeca;
85        cabeca = novoNo;
86        contador++;
87      }
88
89      private No PegarNoNaPosicao(int posicao)
90      {
91        if (posicao < 0 || posicao >= contador)
92        {
93          throw new ArgumentOutOfRangeException(nameof(posicao));
94        }
95
96        No atual = cabeca;
97        for (int i = 0; i < posicao; i++)
98        {
99          atual = atual.Proximo;
100       }
101       return atual;
102     }
103   }
```

**4-)** Escreva um algoritmo para reconhecer se uma dada palavra é um palíndromo.

Considere que a palavra está contida em uma lista simplesmente encadeada, onde cada caractere está em um nó da lista.

```
1   public class No
2   {
3     public char Valor { get; set; }
4     public No Proximo { get; set; }
5
6     public No(char value)
7     {
8       Valor = value;
9       Proximo = null;
10    }
```

```
11 }
12
13 public class ListaEncadeada
14 {
15   private No cabeca;
16   private int contador;
17
18   public ListaEncadeada()
19   {
20     cabeca = null;
21     contador = 0;
22   }
23
24   public void Inserir(char value)
25   {
26     No novoNo = new No(value);
27     if (cabeca == null)
28     {
29       cabeca = novoNo;
30     }
31     else
32     {
33       No atual = cabeca;
34       while (atual.Proximo != null)
35       {
36         atual = atual.Proximo;
37       }
38       atual.Proximo = novoNo;
39     }
40     contador++;
41   }
42
43   public bool EPalindromo()
44   {
45     No lenta = cabeca;
46     No rapida = cabeca;
47     Stack<char> pilha = new Stack<char>();
48
49     while (rapida != null && rapida.Proximo != null)
50     {
51       pilha.Push(lenta.Valor);
52       lenta = lenta.Proximo;
53       rapida = rapida.Proximo.Proximo;
54     }
55
56     if (rapida != null)
57     {
58       lenta = lenta.Proximo;
59     }
```

```
60
61      while (lenta != null)
62      {
63        if (pilha.Pop() != lenta.Valor)
64        {
65          return false;
66        }
67        lenta = lenta.Proximo;
68      }
69
70      return true;
71    }
72  }
```

**5-)** Seja A uma matriz esparsa n x m.

**a-)** Crie uma estrutura de dados que represente A e cujo espaço total seja O(k) em vez de O(mn), onde k é o número total de elementos não irrelevantes de A.

**b-)** Faça um algoritmo para localizar um valor $a_{ij}$ na estrutura acima.

**c-)** Faça um algoritmo para computar $A^2$ utilizando a estrutura acima. Para tal, crie os algoritmos de inserção e busca na sua estrutura.

**d-)** Qual a complexidade da sua solução para a letra (c)?

**R:** O($k^2$), onde k é o número total de elementos não irrelevantes da matriz esparsa.

**5.5-)** Caso não tenha pensado numa estrutura com duas listas (uma para colunas e uma para linhas), recomenda-se refazer o exercício acima com essa forma de estruturação.

**Solução para letras a-), b-) e c-):**

```
1  public class MatrizEsparsa
2  {
3    // Estrutura de dados para representar a matriz esparsa
```

```
4   private List<int> linhas;
5   private List<int> colunas;
6   private List<int> valores;
7
8   // Construtor para inicializar a matriz esparsa
9   public MatrizEsparsa()
10  {
11    linhas = new List<int>();
12    colunas = new List<int>();
13    valores = new List<int>();
14  }
15
16  // a) Metodo para inserir um valor na matriz
17  public void Inserir(int linha, int coluna, int valor)
18  {
19    linhas.Add(linha);
20    colunas.Add(coluna);
21    valores.Add(valor);
22  }
23
24  // b) Metodo para localizar um valor na matriz
25  public int Procurar(int linha, int coluna)
26  {
27    for (int i = 0; i < linhas.Count; i++)
28    {
29      if (linhas[i] == linha && colunas[i] == coluna)
30      {
31        return valores[i];
32      }
33    }
34    return 0; // Valor irrelevante se nao encontrado
35  }
36
37  // c) Metodo para calcular A^2 utilizando a estrutura
38  public MatrizEsparsa Quadrado()
39  {
40    MatrizEsparsa result = new MatrizEsparsa();
41    for (int i = 0; i < linhas.Count; i++)
42    {
43      int linhaAtual = linhas[i];
44      int colunaAtual = colunas[i];
45      int valorAtual = valores[i];
46      for (int j = 0; j < linhas.Count; j++)
47      {
48        if (colunaAtual == linhas[j])
49        {
50          int proximaColuna = colunas[j];
51          int proximoValor = valores[j];
52          int produto = valorAtual * proximoValor;
```

```
53              result.Inserir(linhaAtual, proximaColuna, produto);
54            }
55          }
56        }
57        return result;
58      }
59
60      // Metodo para imprimir a matriz esparsa
61      public void Imprimir()
62      {
63        for (int i = 0; i < linhas.Count; i++)
64        {
65          Console.WriteLine("({0}, {1}): {2}",
66                                      linhas[i],
67                                     colunas[i],
68                                    valores[i]);
69        }
70      }
71  }
```

**Código para execução de testes dos códigos da lista de exercícios**

```
1  Console.WriteLine("Codigos da Lista 01 de exercicios de AED");
2
3  //Questao 01 - Letra A - Exemplos de uso
4
5  var q01A = new Lista01Code.Questao01.LetraA
6                  .ListaDuplamenteEncadeada();
7
8  q01A.Inserir(10);
9  q01A.Inserir(5);
10 q01A.Inserir(6);
11 q01A.Inserir(55);
12
13 var procurar01AChave6 = q01A.Contido(6);
14
15 Console.WriteLine(procurar01AChave6);
16
17 //Remove a chave 6
18 q01A.Remover(6);
19
20 procurar01AChave6 = q01A.Contido(6);
21
22 Console.WriteLine(procurar01AChave6);
23
24
```

```
25 //Questao 01 - Letra B - Exemplos de uso
26
27 var q01B = new Lista01Code.Questao01.LetraB.Fila(); //FIFO
28
29 q01B.Inserir(10);
30 q01B.Inserir(5);
31 q01B.Inserir(6);
32 q01B.Inserir(55);
33
34 var procurar01BChave10 = q01B.Contido(10);
35
36 Console.WriteLine(procurar01BChave10);
37
38
39 //Remove a chave 10
40 q01B.Remover();
41
42 procurar01BChave10 = q01B.Contido(10);
43 Console.WriteLine(procurar01BChave10);
44
45
46 //Questao 01 - Letra C - Exemplo de uso
47
48 var q01C = new Lista01Code.Questao01.LetraC.Pilha(); //LIFO
49
50 q01C.Inserir(10);
51 q01C.Inserir(5);
52 q01C.Inserir(6);
53 q01C.Inserir(55);
54
55 var procurar01CChave55 = q01C.Contido(55);
56
57 Console.WriteLine(procurar01CChave55);
58
59
60 //Remove a chave 55
61 q01C.Remover();
62
63 procurar01CChave55 = q01C.Contido(55);
64 Console.WriteLine(procurar01CChave55);
65
66 //Questao 01 - Letra D - Exemplo de uso
67
68 var q01D = new Lista01Code.Questao01.LetraD
69                 .ListaDuplamenteEncadeadaCircular();
70
71 q01D.Inserir(10);
72 q01D.Inserir(5);
73 q01D.Inserir(6);
```

```
74  q01D.Inserir(55);
75
76  var procurar01DChave55 = q01D.Contido(55);
77
78  Console.WriteLine(procurar01DChave55);
79
80  //Remove a chave 55
81  q01D.Remover(55);
82
83  procurar01DChave55 = q01D.Contido(55);
84  Console.WriteLine(procurar01DChave55);
85
86
87  //Questao 02 - Exemplo de uso
88
89  var q02Lista1 = new Lista01Code.Questao02.ListaInteiroGrande();
90
91  var q02Lista2 = new Lista01Code.Questao02.ListaInteiroGrande();
92
93  var q02Lista3 = new Lista01Code.Questao02.ListaInteiroGrande();
94
95  q02Lista1.InserirDigito(9);
96  q02Lista1.InserirDigito(9);
97  q02Lista1.InserirDigito(9);
98
99  q02Lista2.InserirDigito(5);
100 q02Lista2.InserirDigito(1);
101 q02Lista2.InserirDigito(2);
102
103 //Somar as listas - Resultado esperado 1511 = (999 + 512)
104 q02Lista3 = q02Lista3.Soma(q02Lista1, q02Lista2);
105 q02Lista3.Imprimir();
106
107
108 //Questao 03 - Exemplo de uso
109 var q03 = new Lista01Code.Questao03.ListaEncadeada();
110
111 q03.Inserir(10);
112 q03.Inserir(5);
113 q03.Inserir(6);
114 q03.Inserir(55);
115
116 q03.Imprimir();
117
118 q03 = q03.InverterLista();
119
120 q03.Imprimir();
121
122 var q03Soma = new Lista01Code.Questao03.ListaEncadeada();
```

```
123 |
124 | q03Soma = q03.SomarMeiaLista();
125 |
126 | q03Soma.Imprimir();
127 |
128 | //Questao 04 - Exemplo de uso
129 |
130 | var q0401 = new Lista01Code.Questao04.ListaEncadeada();
131 |
132 | q0401.Inserir('R');
133 | q0401.Inserir('A');
134 | q0401.Inserir('D');
135 | q0401.Inserir('A');
136 | q0401.Inserir('R');
137 |
138 | Console.WriteLine(q0401.EPalindromo() ?
139 |                       "E palindromo" :
140 |                   "Nao e Palindromo");
141 |
142 | var q0402 = new Lista01Code.Questao04.ListaEncadeada();
143 |
144 | q0402.Inserir('B');
145 | q0402.Inserir('A');
146 | q0402.Inserir('T');
147 | q0402.Inserir('A');
148 | q0402.Inserir('T');
149 | q0402.Inserir('A');
150 |
151 | Console.WriteLine(q0402.EPalindromo() ?
152 |                       "E palindromo" :
153 |                   "Nao e Palindromo");
154 |
155 |
156 | //Questao 05 - Exemplo de uso
157 |
158 | var q05 = new Lista01Code.Questao05.MatrizEsparsa();
159 |
160 | // a) Inserindo valores na matriz
161 | q05.Inserir(0, 0, 2);
162 | q05.Inserir(0, 1, 3);
163 | q05.Inserir(0, 2, 4);
164 | q05.Inserir(1, 0, 5);
165 | q05.Inserir(1, 1, 6);
166 | q05.Inserir(1, 2, 7);
167 |
168 | Console.WriteLine("Matriz Esparsa:");
169 | q05.Imprimir();
170 |
171 |
```

```
172 // b) Localizando um valor na matriz
173 int valor = q05.Procurar(1, 0);
174 Console.WriteLine("Valor encontrado: " + valor);
175
176 // c) Calculando A^2
177 var q05Quadrado = q05.Quadrado();
178 Console.WriteLine("Matriz A^2:");
179 q05Quadrado.Imprimir();
180
181 // d) complexidade O(k^2) do medodo Quadrado(), sendo K
182 o numero de elementos nao irrelevantes.
183
184 Console.ReadKey();
```