

AED1 – Aula 6

Listas

Prof. Dr. Aldo Díaz
aldo.diaz@ufg.br

Instituto de Informática
Universidade Federal de Goiás

08/12/2022

- 1 Introdução
- 2 Operações fundamentais
- 3 Representações
- 4 Listas Lineares Encadeadas
- 5 Lista Linear Simplesmente Encadeada

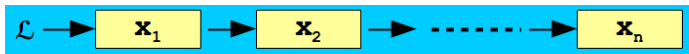
Introdução

Há problemas que implicam uma **relação linear de ordem** entre os elementos de um conjunto: $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \rightarrow x_n$.

Exemplos: Lista de compras de supermercado, tarefas da semana, e-mails a serem lidos/respondidos, rotina de exercícios na academia.

Para a **solução**, é necessário desenvolver uma estrutura de dados adequada chamada de **lista linear**, representada por:

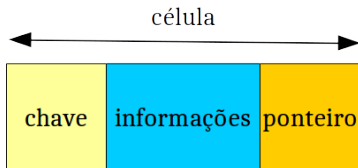
$$\mathcal{L} = (x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \rightarrow x_{n-1} \rightarrow x_n)$$



Conceito

Cada célula da lista é uma **estrutura** que contém:

- Uma **chave primária**, ou identificador único da célula na lista.
- Um conjunto de **dados**, representando uma informação que se deseja armazenar.



Operações fundamentais

Que operações se pode realizar numa lista linear?

- a. Criar lista
- b. Inserir uma célula antes/depois de determinada célula
- c. Quantificar as células presentes na lista
- d. Consultar a célula que possui certa chave
- e. Consultar a célula que está na k -ésima posição

Operações fundamentais

Habitualmente as operações fundamentais sobre listas lineares incluem:

- f. Remover a célula que possui certa chave
- g. Remover a célula que está na k -ésima posição
- h. Ordenar as células da lista
- i. Concatenar listas
- j. Inverter a ordem das células de uma lista
- k. Comparar listas (verificar se são iguais)

Operações fundamentais

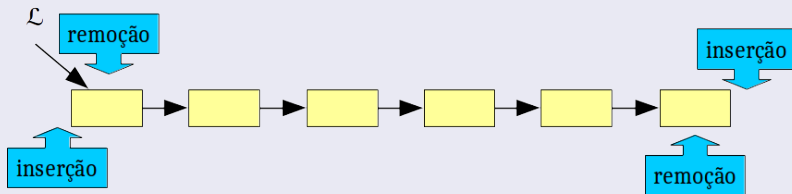
Aplicando-se algumas *restrições* às operações permitidas surgem 3 *variações* de listas lineares:

1. Deque
2. Fila
3. Pilha

Operações fundamentais

1. Deque

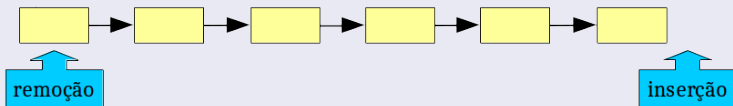
As inserções e remoções são permitidas em ambas extremidades:



Operações fundamentais

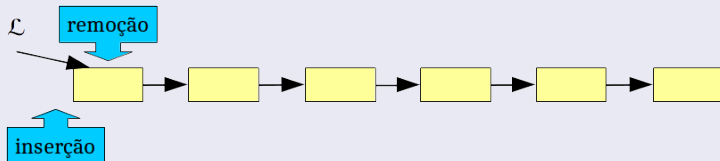
2. Fila

As inserções são permitidas numa extremidade e as remoções na outra (política FIFO).



3. Pilha

As inserções e remoções são permitidas numa única extremidade (política LIFO).



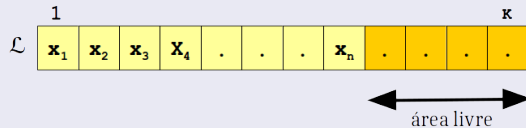
Representações

Para implementação computacional de uma lista linear há, essencialmente, duas representações:

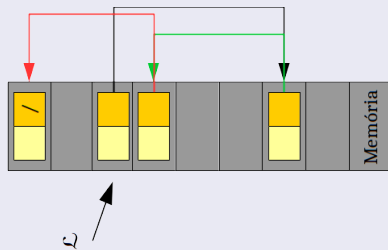
1. Por contiguidade (alocação estática)
2. Por encadeamento (alocação dinâmica)

Representações

1. Alocação estática



2. Alocação dinâmica



Representações

1. Alocação estática

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define SUCESSO 1
5 #define FALHA -1
6 #define CHAVE_INVALIDA 0
7 #define TAMANHO_MAXIMO_LISTA 100
8
9 // Declaracao de estruturas
10 typedef struct {
11     unsigned int chave;
12     unsigned int dado;
13 } Celula;
14
15 typedef struct {
16     Celula celulas[TAMANHO_MAXIMO_LISTA];
17     unsigned int tamanho;
18 } ListaLinear;
```

Representações

1. Alocação estática

```
1 // Criar uma lista inicialmente vazia
2 int criarListaVazia(ListaLinear *lista) {
3     lista->tamanho = 0;
4
5     return(SUCESSO);
6 }
7
8 // Ou, com inicializacao das celulas da lista com chaves invalidas
9 int criarListaVazia(ListaLinear *lista) {
10     for(int i=0; i<TAMANHO_MAXIMO_LISTA; i++)
11         lista->celulas[i].chave = CHAVE_INVALIDA;
12
13     lista->tamanho = 0;
14
15     return(SUCESSO);
16 }
```

Representações

1. Alocação estática

```
1 // Criar uma lista com uma única célula
2 int criarListaChave(ListaLinear *lista, Celula celula) {
3     lista->celulas[0] = celula;
4     lista->tamanho = 1;
5
6     return(SUCESSO);
7 }
8
9 // Ou, inicializando as demais células
10 int criarListaChave(ListaLinear *lista, Celula celula) {
11     for(int i=0; i<TAMANHO_MAXIMO_LISTA; i++)
12         lista->celulas[i].chave = CHAVE_INVALIDA;
13
14     lista->celulas[0] = celula;
15     lista->tamanho = 1;
16
17     return(SUCESSO);
18 }
```

Representações

1. Alocação estática

```
1 // Determinar o tamanho da lista
2 int tamanhoLista(ListaLinear lista) {
3     if(lista.tamanho >= 0)
4         return(lista.tamanho);
5     else
6         return(FALHA);
7 }

1 // Mostrar uma determinada célula da lista
2 void mostrarCelula(Celula celula) {
3     printf("Chave : %u\n", celula.chave);
4     printf("Dado : %u\n", celula.dado);
5 }
```

Representações

1. Alocação estática

```
1 // Mostrar toda a lista
2 void mostrarLista(ListaLinear lista) {
3     if(lista.tamanho == 0)
4         printf("Atencao: a lista esta vazia.\n");
5     else{
6         printf("A lista linear possui %u elementos.\n", lista.tamanho);
7         for(int i=0; i<lista.tamanho; i++) {
8             printf("Elemento n.: %u\n", (i+1));
9             mostrarCelula(lista.celulas[i]);
10        }
11    }
12 }
```


Representações

1. Alocação estática

```
1 // Consultar célula baseado na posição dela
2 Celula consultaListaPosicao(ListaLinear lista, unsigned int intPosicao) {
3     Celula celulaResultado;
4
5     if((intPosicao > 0) && (intPosicao <= lista.tamanho))
6         celulaResultado = lista.celulas[intPosicao - 1];
7     else
8         celulaResultado.chave = CHAVE_INVALIDA;
9
10    return(celulaResultado);
11 }
```

Representações

1. Alocação estática

```
1 // Consultar célula baseado na chave dela
2 Celula consultaListaChave(ListaLinear lista, Celula celula) {
3     for(int i=0; i<lista.tamanho; i++)
4         if(lista.celulas[i].chave == celula.chave)
5             return(lista.celulas[i]);
6
7     celula.chave = CHAVE_INVALIDA;
8
9     return(celula);
10 }
```

Representações

1. Alocação estática

```
1 // Inserir célula no início da lista
2 int insInicio(ListaLinear *lista, Celula celula) {
3     if(lista->tamanho == TAMANHO_MAXIMO_LISTA)
4         return(FALHA); // Lista cheia: Overflow!
5     else {
6         for(int i=lista->tamanho; i>0; i--)
7             lista->celulas[i] = lista->celulas[i-1];
8
9         lista->celulas[0] = celula; // Insercao no inicio da lista
10        lista->tamanho++;
11
12        return(SUCESSO);
13    }
14 }
```

Representações

1. Alocação estática

```
1 // Inserir celula no final da lista
2 int insFinal(ListaLinear *lista, Celula celula) {
3     Celula auxiliar;
4
5     if(lista->tamanho == TAMANHO_MAXIMO_LISTA)
6         return(FALHA); // Lista cheia: Overflow!
7     else{
8         lista->celulas[lista->tamanho] = celula; // Insercao no final da lista
9         lista->tamanho++;
10
11         return(SUCESSO);
12     }
13 }
```

Representações

1. Alocação estática

```
1 // Inserir celula por chave (ascendente)
2 int insOrdem(ListaLinear *lista, Celula celula) {
3     int i;
4
5     if(lista->tamanho == TAMANHO_MAXIMO_LISTA)
6         return(FALHA); // Lista cheia: Overflow!
7     else
8         if(lista->tamanho == 0)
9             return(insInicio(lista, celula));
10        else
11            if(celula.chave < lista->celulas[0].chave)
12                return(insInicio(lista, celula));
13            else
14                if(celula.chave >= lista->celulas[lista->tamanho-1].chave)
15                    return(insFinal(lista, celula));
16                else {
```

Representações

1. Alocação estática

```
1      // Continuacao de insOrdem
2      i = 0;
3
4      while((celula.chave >= lista->celulas[i].chave) && (i < lista->tamanho))
5          i++;
6
7      if(i == lista->tamanho)
8          return(insFinal(lista, celula));
9      else {
10         for(int j=lista->tamanho; j>i; j--)
11             lista->celulas[j] = lista->celulas[j-1];
12
13         lista->celulas[i] = celula;
14         lista->tamanho++;
15
16         return(SUCESSO);
17     }
18 }
19 }
```

Representações

1. Alocação estática

```
1 // Remover célula no início da lista
2 Celula remInicio(ListaLinear *lista) {
3     Celula celulaResultado;
4
5     if(lista->tamanho == 0) {
6         celulaResultado.chave = CHAVE_INVALIDA;
7
8         return(celulaResultado);
9     }
10    else {
11        celulaResultado = lista->celulas[0];
12
13        for(int i=0; i<lista->tamanho-1; i++)
14            lista->celulas[i] = lista->celulas[i+1];
15
16        lista->tamanho--;
17
18        return(celulaResultado);
19    }
20 }
```

Representações

1. Alocação estática

```
1 // Remover célula no final da lista
2 Celula remFinal(ListaLinear *lista) {
3     Celula celulaResultado;
4
5     if(lista->tamanho == 0) {
6         celulaResultado.chave = CHAVE_INVALIDA;
7
8         return(celulaResultado);
9     }
10    else {
11        celulaResultado = lista->celulas[lista->tamanho-1];
12        lista->tamanho--;
13
14        return(celulaResultado);
15    }
16 }
```


Representações

1. Alocação estática

```

1  // Remover celula baseado na chave
2  Celula remChave(ListaLinear *lista, Celula celula) {
3      unsigned int i, j, k;
4      unsigned int intQuantidadeRemocoes;
5
6      Celula celulaResultado;
7
8      if(lista->tamanho == 0) {
9          celulaResultado.chave = CHAVE_INVALIDA;
10
11         return(celulaResultado);
12     }
13     else
14         if(celula.chave == lista->celulas[0].chave)
15             while(celula.chave == lista->celulas[0].chave) {
16                 celulaResultado = remInicio(lista);
17
18                 if(celulaResultado.chave == CHAVE_INVALIDA)
19                     return(celulaResultado);
20             }
21         else
22             if(celula.chave == lista->celulas[lista->tamanho-1].chave)
23                 while(celula.chave == lista->celulas[lista->tamanho-1].chave) {
24                     celulaResultado = remFinal(lista);

```

Representações

1. Alocação estática

```

1      // Continuacao de remChave
2      if(celulaResultado.chave == CHAVE_INVALIDA)
3          return(celulaResultado);
4      }
5      else {
6          i = 0;
7
8          while((celula.chave > lista->celulas[i].chave) && (i < lista->tamanho))
9              i++;
10
11         if(i == lista->tamanho) {
12             celulaResultado.chave = CHAVE_INVALIDA;
13
14             return(celulaResultado);
15         }
16         else {
17             intQuantidadeRemocoes = 0;
18             j = i;
19
20             while((celula.chave == lista->celulas[j].chave) && (j < lista->tamanho)) {
21                 intQuantidadeRemocoes++;
22                 j++;
23             }

```

Representações

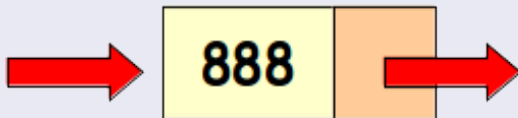
1. Alocação estática

```
1      // Continuacao de remChave
2      if(intQuantidadeRemocoes == 0) {
3          celulaResultado.chave = CHAVE_INVALIDA;
4
5          return(celulaResultado);
6      }
7      else {
8          celulaResultado = lista->celulas[i];
9
10         for(j=i; j<(lista->tamanho - intQuantidadeRemocoes); j++)
11             if(j + intQuantidadeRemocoes < lista->tamanho)
12                 lista->celulas[j] = lista->celulas[j + intQuantidadeRemocoes];
13
14         lista->tamanho -= intQuantidadeRemocoes;
15
16         return(celulaResultado);
17     }
18 }
19 }
20 }
```

Representações

2. Lista por encadeamento (alocação dinâmica)

Uma lista linear $\mathcal{L} = (x_1, x_2, \dots, x_n)$ será representada por uma sequência de células, onde cada uma delas **aponta para** a sua sucessora.



Listas lineares encadeadas

Ao utilizarmos a representação **por encadeamento** é possível implementar diversas variações de listas lineares:

1. Simplesmente encadeada (LLSE)
2. Simplesmente encadeada com nó descritor (LLSEcD)
3. Duplamente encadeada (LLDE)
4. Duplamente encadeada com nó descritor (LLDEcD)

Listas lineares encadeadas

Há, adicionalmente, a possibilidade de as tornarmos **listas circulares**:

6. Circular simplesmente encadeada (LLCSE)
7. Circular simplesmente encadeada com nó descritor (LLCSEcD)
8. Circular duplamente encadeada (LLCDE)
9. Circular duplamente encadeada com nó descritor (LLCDEcD)

Lista linear simplesmente encadeada

Definição

Uma lista linear $\mathcal{L}_1 = (x_1, x_2, \dots, x_n)$ é dita **simplesmente encadeada** (LLSE) quando se utiliza uma estrutura que possui apenas 1 ponteiro indicando a célula sucessora.

Dessa forma, cada célula terá os seguintes membros:

- **chave**: chave primária de cada célula
- **dados**: representa o conteúdo relevante, dados, a serem armazenados naquela célula
- **prox**: ponteiro que indica o endereço da próxima célula na lista ou NULL quando não houver próxima.

Lista linear simplesmente encadeada

Operações fundamentais

É necessário implementar algumas **operações fundamentais**:

1. Criar uma lista vazia
2. Criar uma lista com uma célula com um certo par (**chave, dado**)

Lista linear simplesmente encadeada

Operações fundamentais

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Constantes para uso geral
5 #define SUCESSO 1
6 #define FALHA -1
7 #define CHAVE_INVALIDA 0
8
9 typedef struct Celula *ApontadorCelula;
10 typedef struct Celula {
11     int chave;
12     int dado;
13     ApontadorCelula prox;
14 } Celula;
```

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Criar lista vazia
2 int CriarListaVazia(ApontadorCelula *p) {
3     (*p) = (ApontadorCelula *) NULL;
4
5     return(SUCESSO);
6 }
```

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Criar lista com uma celula inicial
2 int CriarListaChave(ApontadorCelula *p, Celula celula) {
3     int statusOperacao;
4
5     (*p) = (ApontadorCelula *) NULL;
6     statusOperacao = InsInicio(p, celula);
7
8     return(statusOperacao);
9 }
10
11 // Ou ...
12 int CriarListaChave(ApontadorCelula *p, Celula celula) {
13     (*p) = (ApontadorCelula *) NULL;
14
15     return(InsInicio(p, celula));
16 }
```

Lista linear simplesmente encadeada

Operações Fundamentais

É necessário implementar algumas **operações fundamentais**:

3. Inserir um certo par (**chave, dado**) no início da lista
4. Inserir um certo par (**chave, dado**) no final da lista

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Inserir uma celula no inicio da lista
2 int InsInicio(ApontadorCelula *p, Celula celula) {
3     ApontadorCelula q;
4
5     q = (ApontadorCelula) malloc(sizeof(Celula));
6     if(q == NULL)
7         return (FALHA);
8     else {
9         q->chave = celula.chave;
10        q->dado = celula.dado;
11        q->prox = (ApontadorCelula) (*p);
12        (*p) = q;
13
14        return (SUCESSO);
15    }
16 }
```

Lista linear simplesmente encadeada

Operações fundamentais

```

1 // Inserir uma celula no final da lista
2 int InsFinal (ApontadorCelula *p, Celula celula) {
3     ApontadorCelula q, r;
4
5     if((*p) == NULL)
6         return(InsInicio (p, celula));
7     else {
8         q = (Ponteiro) malloc(sizeof(Celula));
9
10        if(q == NULL)
11            return (FALHA);
12        else {
13            q->chave = celula.chave;
14            q->dado = celula.dado;
15            q->prox = (ApontadorCelula) NULL;
16            r = (*p);
17            while(r->prox != NULL)
18                r = r->prox;
19            r->prox = q;
20
21            return (SUCESSO);
22        }
23    }
24 }

```

Lista linear simplesmente encadeada

Operações fundamentais

É necessário implementar algumas **operações fundamentais**:

5. Inserir um certo par (**chave, dado**) de tal maneira que as chaves da lista se mantenham em ordem crescente (ou decrescente)
6. Remover a célula do início da lista

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Inserir uma celula em ordem
2 int InsOrdem (ApontadorCelula * p, Celula celula) {
3     ApontadorCelula q, r, s;
4
5     if((*p) == NULL)
6         return(InsInicio(p, celula));
7     else
8         if((*p)->prox == NULL) {
9             if(celula.chave < (*p)->chave)
10                 return(InsInicio(p, celula));
11             else
12                 return(InsFinal(p, celula));
13         else
14             if(celula.chave < (*p)->chave)
15                 return(InsInicio(p, celula));
16             else {
17                 q = (ApontadorCelula) malloc(sizeof(Celula));
18
19                 if(q == NULL)
20                     return(FALHA);
21
22                 q->chave = celula.chave;
23                 q->dado = celula.dado;
24                 r = (*p);
```


Lista linear simplesmente encadeada

Operações fundamentais

```
1      // Continuação de insOrdem
2      while(((r->chave) < celula.chave) && ((r->prox) != NULL)) {
3          s = r;
4          r = r->prox;
5      }
6
7      if(r->chave <= celula.chave) {
8          q->prox = (ApontadorCelula) NULL;
9          r->prox = q;
10     }
11     else {
12         q->prox = r;
13         s->prox = q;
14     }
15 }
16
17
18 return(SUCESSO);
19 }
```

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Remover uma celula do início da lista
2 int RemInicio (ApontadorCelula *p) {
3     ApontadorCelula r;
4
5     if((*p) == NULL)
6         return (FALHA);
7     else {
8         r = (*p);
9         (*p) = (*p)->prox;
10        free(r);
11
12        return(SUCESSO);
13    }
14 }
```

Lista linear simplesmente encadeada

Operações fundamentais

É necessário implementar algumas **operações fundamentais**:

7. Remover a célula que possui a chave **chave** da lista

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Remover uma célula com certa chave
2 int RemChave (ApontadorCelula *p, Celula celula) {
3     ApontadorCelula r, s;
4
5     if((*p) == NULL)
6         return(FALHA);
7     else {
8         r = (*p);
9         s = (*p);
10
11         if((r->prox) == NULL)
12             if(r->chave == celula.chave)
13                 return(RemInicio(p));
14             else
15                 return(FALHA);
```

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Continuacao de remChave
2 else
3     if(r->chave == celula.chave)
4         return(RemInicio(p));
5     else {
6         while(((r->prox) != NULL) && (r->chave != celula.chave)) {
7             s = r;
8             r = r->prox;
9         }
10
11         if(r->chave == celula.chave) {
12             s->prox = r->prox;
13             free(r);
14
15             return(SUCESSO);
16         }
17         else
18             return(FALHA);
19     }
20 }
21 }
```

Lista linear simplesmente encadeada

Operações fundamentais

É necessário implementar algumas **operações fundamentais**:

8. Remover a célula que está no final da lista

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Remover uma celula do final da lista
2 int RemFinal(ApontadorCelula *p) {
3     ApontadorCelula r, s;
4
5     if((*p) == NULL)
6         return(FALHA);
7     else {
8         r = (*p);
9         s = (*p);
10
11         if((r->prox) == NULL)
12             return(RemInicio(p));
13         else {
14             while((r->prox) != NULL) {
15                 s = r;
16                 r = r->prox;
17             }
18
19             s->prox = (ApontadorCelula) NULL;
20             free(r);
21
22             return(SUCESSO);
23         }
24     }
25 }
```

Lista linear simplesmente encadeada

Operações fundamentais

É necessário implementar algumas **operações fundamentais**:

9. Determinar o número de células presente na lista
10. Concatenar duas listas: \mathcal{L}_1 e \mathcal{L}_2

Lista linear simplesmente encadeada

Operações fundamentais

```
1 // Determinar o tamanho da lista linear
2 int TamLista(ApontadorCelula p) {
3     ApontadorCelula r;
4     int tam;
5
6     tam = 0;
7     if(p == NULL)
8         return(tam);
9     else {
10        tam = 1;
11        r = p;
12        while(r->prox != NULL) {
13            ++tam;
14            r = r->prox;
15        }
16
17        return(tam);
18    }
19 }
```

Lista linear simplesmente encadeada


Operações fundamentais

```
1 // Concatenar duas listas gerando uma nova lista
2 // (que deve ser considerada, inicialmente, nao inicializada)
3 // Tanto p quanto q pode, inicialmente, estar vazia.
4 int ConcatListas(ApontadorCelula p, ApontadorCelula q, ApontadorCelula *lista) {
5     // Para o estudante elaborar ...
6
7     return(SUCESSO);
8 }
```

Saiba mais ...


youtu.be

- Canal do [Prof. Wanderley de Souza Alencar](#) (INF/UFG)




Wanderley de Souza Alencar
 @wanderleydesouzaalencar7933
 329 subscribers


HOME VIDEOS PLAYLISTS COMMUNITY CHANNELS ABOUT



Listas Lineares 1ª Parte
 per Wanderley de Souza Alencar
 @wanderleydesouzaalencar7933
 535 views • 2 years ago
 Videoaula sobre "Listas Lineares" para os(as) estudantes dos cursos de bacharelado do Instituto de Informática da Universidade Federal de Goiás, período noturno do 1º semestre letivo de...
 September 27, 2020 **16:48**



Listas Lineares Simplesmente Encadeadas com Nó Descritor
 Wanderley de Souza Alencar • 225 views • 4 years ago
 Aula sobre listas lineares simplesmente encadeadas com nó descritor (LLSECD) para estudantes da disciplina Algoritmos e Estruturas de Dados - 1 do Curso de Bacharelado em Sistemas de Informação...
17:13



Listas Lineares Simplesmente Encadeadas (LLSE)
 Wanderley de Souza Alencar • 477 views • 4 years ago
 Aula sobre Listas Lineares Simplesmente Encadeadas (LLSE) para a turma da disciplina "Algoritmos e Estruturas de Dados - 1" do Curso de Bacharelado em Sistemas de Informação, Universidade...
1:08:28