

AED1 – Aula 7

Listas LLDE

Prof. Dr. Aldo Díaz
aldo.diaz@ufg.br

Instituto de Informática
Universidade Federal de Goiás

INF

INSTITUTO DE
INFORMÁTICA



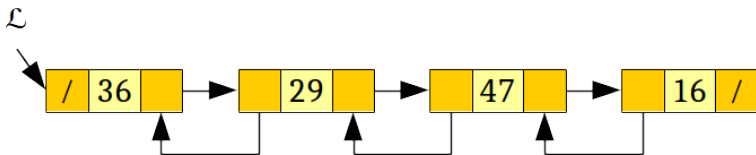
UFG

UNIVERSIDADE
FEDERAL DE GOIÁS

15/12/2022

1 Lista linear duplamente encadeada

Lista linear duplamente encadeada



Lista linear duplamente encadeada

Uma **lista linear duplamente encadeada** (LLDE) utiliza, além dos dados da célula, 2 ponteiros indicando as células precedente e sucessora, respectivamente.

Dessa forma, a representação de cada célula terá os seguintes membros:

- a. anterior: Apontador para o endereço da célula precedente na lista, ou NULL quando não houver célula precedente.
- b. chave: Chave primária de cada célula.
- c. dados: O conteúdo relevante, os dados a serem armazenados naquela célula.
- d. proximo: Apontador para o endereço da célula sucessora na lista, ou NULL quando não houver célula sucessora.

Lista linear duplamente encadeada

Operações fundamentais

É necessário implementar algumas **operações fundamentais**:

1. Criar uma lista vazia
2. Criar uma lista com uma célula inicial
3. Inserir um par (chave, dado) no início da lista
4. Inserir um par (chave, dado) no final da lista
5. Inserir um par (chave, dado) mantendo as chaves da lista em ordem (crescente, ou decrescente)
6. Remover a célula com uma certa chave da lista
7. Quantificar as células presentes na lista
8. Concatenar duas listas \mathcal{L}_1 e \mathcal{L}_2

Lista linear duplamente encadeada

llde.h

```
1 // Declaracao de estruturas
2 typedef struct Celula* ApontadorCelula;
3 typedef struct Celula Celula;
4
5 struct Celula {
6     ApontadorCelula ant;
7     int chave;
8     int dado;
9     ApontadorCelula prox;
10 };
11
12 ApontadorCelula prim;
```

Lista linear duplamente encadeada

llde.h

```
1 // Prototipos das operacoes
2 void LLDEcriarLayoutVazia(ApontadorCelula *p);
3 void LLDEcriarLayoutChave(ApontadorCelula *p, Celula celula);
4 void LLDEmostrarLista(ApontadorCelula p);
5 void LLDEmostrarListaReversa(ApontadorCelula p);
6 void LLDEinserirInicio(ApontadorCelula *p, Celula celula);
7 void LLDEinserirFinal(ApontadorCelula *p, Celula celula);
8 void LLDEinserirOrdem(ApontadorCelula *p, Celula celula);
9 void LLDERemoverInicio(ApontadorCelula *p);
10 void LLDERemoverFinal(ApontadorCelula *p);
11 void LLDERemoverChave(ApontadorCelula *p, Celula celula);
12 void LLDEconcatenarListas(ApontadorCelula p, ApontadorCelula q, ApontadorCelula *lista);
13 int LLDEtamanhoLista(ApontadorCelula p);
```

Lista linear duplamente encadeada

llde.c

```
1 // Criar lista vazia
2 void LLDEcriarLayoutVazia(ApontadorCelula *p) {
3     (*p) = (ApontadorCelula) NULL;
4 }

1 // Criar lista com uma celula inicial
2 void LLDEcriarLayoutChave(ApontadorCelula *p, Celula celula) {
3     (*p) = (ApontadorCelula) malloc(sizeof(ApontadorCelula *));
4
5     if((*p) == NULL)
6         printf("Erro: Sem memoria disponivel\n");
7     else
8         LLDEinserirInicio(p, celula);
9 }
```


Lista linear duplamente encadeada

llde.c

```
1 // Inserir uma celula no inicio da lista
2 void LLDEinserirInicio(ApontadorCelula *p, Celula celula) {
3     ApontadorCelula q;
4
5     q = (ApontadorCelula) malloc(sizeof(ApontadorCelula *));
6
7     if(q == NULL)
8         printf("Erro: Sem memoria disponivel\n");
9     else {
10         q->chave = celula.chave;
11         q->dado = celula.dado;
12         q->ant = (ApontadorCelula) NULL;
13         q->prox = (*p);
14         (*p) = q;
15     }
16 }
```

Lista linear duplamente encadeada

llde.c

```
1 // Inserir uma celula no final da lista
2 void LLDEinserirFinal(ApontadorCelula *p, Celula celula) {
3     ApontadorCelula q, r;
4
5     if((*p) == NULL)
6         LLDEinserirInicio(p, celula);
7     else {
8         q = (ApontadorCelula) malloc(sizeof(ApontadorCelula *));
9
10        if(q == NULL)
11            printf("Erro: Sem memoria disponivel\n");
12        else {
13            q->chave = celula.chave;
14            q->dado = celula.dado;
15            r = (*p);
16
17            while(r->prox != NULL)
18                r = r->prox;
19            q->ant = r;
20            q->prox = r->prox;
21            r->prox = q;
22        }
23    }
24 }
```

Lista linear duplamente encadeada

llde.c

```
1 // Inserir uma celula em ordem na lista
2 void LLDEinserirOrdem(ApontadorCelula *p, Celula celula) {
3     ApontadorCelula q, r;
4
5     if((*p) == NULL)
6         LLDEinserirInicio(p, celula);
7     else
8         if(celula.chave < ((*p)->chave))
9             LLDEinserirInicio(p, celula);
10        else {
11            q = (ApontadorCelula) malloc(sizeof(ApontadorCelula *));
12
13            if(q == NULL)
14                printf("Erro: Sem memoria disponivel\n");
15            else {
16                /* Bloco 1 */
17            }
18        }
19 }
```

Lista linear duplamente encadeada

llde.c

```
1      /* Bloco 1 */
2      q->chave = celula.chave;
3      q->dado = celula.dado;
4      r = (*p);
5
6      while((r->chave <= q->chave) && (r->prox != NULL))
7          r = r->prox;
8      if(r->prox == NULL)
9          if(q->chave < r->chave) {
10              q->ant = r->ant;
11              q->prox = r;
12              (*(r->ant)).prox = q;
13              r->ant = q;
14          }
15          else {
16              q->ant = r;
17              q->prox = r->prox;
18              r->prox = q;
19          }
20      else {
21          q->ant = r->ant;
22          q->prox = r;
23          (*(r->ant)).prox = q;
24          r->ant = q;
25      }
```

Lista linear duplamente encadeada

llde.c

```
1 // Remover uma celula no inicio
2 void LLDEremoverInicio(ApontadorCelula *p) {
3     ApontadorCelula r;
4
5     if((*p) == NULL)
6         printf("A lista esta VAZIA!\n");
7     else {
8         r = (*p);
9
10        if ((*p)->prox == NULL)
11            (*p) = (ApontadorCelula) NULL;
12        else {
13            (*p) = r->prox;
14            (*p)->ant = (ApontadorCelula) NULL;
15        }
16
17        free(r);
18    }
19 }
```

Lista linear duplamente encadeada

llde.c

```
1 // Remover uma celula no final
2 void LLDEremoverFinal(ApontadorCelula *p) {
3     ApontadorCelula r;
4
5     if((*p) == NULL)
6         printf("A lista esta VAZIA!\n");
7     else {
8         r = (*p);
9
10        if((r->prox == NULL)
11            (*p) = (ApontadorCelula) NULL;
12        else {
13            while(r->prox != NULL)
14                r = r->prox;
15            (*(r->ant)).prox = r->prox;
16        }
17
18        free(r);
19    }
20 }
```

Lista linear duplamente encadeada

llde.c

```
1 // Remover uma celula pela chave
2 void LLDEremoverChave(ApontadorCelula *p, Celula celula) {
3     ApontadorCelula r, n;
4
5     if((*p) == NULL)
6         printf("A lista esta VAZIA!\n");
7     else {
8         r = (*p);
9
10        if(r->prox == NULL)
11            if(r->chave == celula.chave)
12                LLDEremoverInicio(p);
13            else
14                printf("Celula INVALIDA!\n");
15        else {
16            /* Bloco da remocao de chave */
17        }
18    }
19 }
```

Lista linear duplamente encadeada

llde.c

```
1      /* Bloco da remocao da chave */
2      while(r != NULL)
3          if(r->chave == celula.chave)
4              if(r->ant == NULL) {
5                  LLDEremoveInicio(p);
6                  r = (*p);
7              }
8              else {
9                  n = r->prox;
10                 (*(r->ant)).prox = n;
11                 if(n != NULL)
12                     n->ant = r->ant;
13                 free(r);
14                 // r = n;
15             }
16      else
17          r = r->prox;
```


Lista linear duplamente encadeada

llde.c

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

Lista linear duplamente encadeada


llde.c

```
1 // Concatenar duas listas (ambas podem estar inicialmente vazias)
2 void LLDEconcatenarListas(ApontadorCelula p, ApontadorCelula q, ApontadorCelula *lista) {
3     /*
4     Para o estudante elaborar...
5     */
6
7     printf("Listas CONCATENADAS\n");
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
 Universidade Federal de Goiás - UFG
 Instituto de Informática - INF
 September 27, 2020
 16:48

ALGORITMOS E ESTRUTURAS DE DADOS

Aula sobre Listas Lineares (1ª parte)
 Wanderley de Souza Alencar • 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...

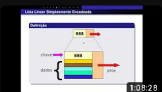


Listas Lineares Simplesmente Encadeadas com Nó Descritor
 Wanderley de Souza Alencar • 225 views • 4 years ago

ALGORITMOS E ESTRUTURAS DE DADOS

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...



Listas Lineares Simplesmente Encadeadas (LLSE)
 Wanderley de Souza Alencar • 477 views • 4 years ago

ALGORITMOS E ESTRUTURAS DE DADOS

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...