

Estruturas de Dados – Aula 04

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Tipo abstrato de dados

- Um **tipo de dado**, em programação, define qual o conjunto de valores (domínio) que uma variável pode armazenar e quais as operações que podem ser realizadas sobre tais valores. Exemplo: int, float, char, ...
- Um **tipo abstrato de dados (TAD)** é um conjunto de dados estruturados com operações sobre esses dados, sem especificação de como essas operações são implementadas. Exemplo: Listas ligadas/encadeadas, pilhas, filas, ...

TAD Ponto

```
1  #include <stdio.h>
2
3  // Definição da estrutura para representar um ponto
4  typedef struct {
5      double x;
6      double y;
7  } Ponto;
8
9  // Função para criar um novo ponto
10 Ponto criarPonto(double x, double y) {
11     Ponto novoPonto;
12     novoPonto.x = x;
13     novoPonto.y = y;
14     return novoPonto;
15 }
16
```

TAD Ponto

```
17 // Função para somar dois pontos
18 Ponto somarPontos(Ponto p1, Ponto p2) {
19     Ponto P;
20     P.x = p1.x + p2.x;
21     P.y = p1.y + p2.y;
22     return P;
23 }
24
25 // Função para imprimir as coordenadas de um ponto
26 void imprimirPonto(Ponto p) {
27     printf("(%.21f, %.21f)\n", p.x, p.y);
28 }
29
```

TAD Ponto

```
30 int main() {  
31     // Exemplo de uso do TAD Ponto  
32     Ponto A = criarPonto(3.0, 4.0);  
33     Ponto B = criarPonto(6.0, 8.0);  
34  
35     printf("Coordenadas do ponto A = ");  
36     imprimirPonto(A);  
37  
38     printf("Coordenadas do ponto B = ");  
39     imprimirPonto(B);  
40  
41     Ponto S = somarPontos(A, B);  
42  
43     printf("\nA + B = ");  
44     imprimirPonto(S);  
45  
46     return 0;  
47 }
```

TAD Ponto

```
Coordenadas do ponto A = (3.00, 4.00)  
Coordenadas do ponto B = (6.00, 8.00)  
  
A + B = (9.00, 12.00)
```

TAD Ponto (com ponteiro)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Definição da estrutura para representar um ponto
5  typedef struct {
6      double x;
7      double y;
8  } Ponto;
9
10 typedef Ponto *Ponteiro; // Ponteiro para a estrutura Ponto
11
12 // Função para criar um novo ponto
13 Ponteiro criarPonto(double x, double y) {
14     Ponteiro novoPonto = (Ponteiro)malloc(sizeof(Ponto));
15     novoPonto->x = x;
16     novoPonto->y = y;
17     return novoPonto;
18 }
19
```

TAD Ponto (com ponteiro)

```
20 // Função para somar dois pontos
21 Ponteiro somarPontos(Ponteiro p1, Ponteiro p2) {
22     Ponteiro P = criarPonto(p1->x + p2->x, p1->y + p2->y);
23     return P;
24 }
25
26 // Função para imprimir as coordenadas de um ponto
27 void imprimirPonto(Ponteiro p) {
28     printf("(%.21f, %.21f)\n", p->x, p->y);
29 }
30
```


TAD Ponto (com ponteiro)

```
31 int main() {  
32     // Exemplo de uso do TAD Ponto  
33     Ponto pA = criarPonto(3.0, 4.0);  
34     Ponto pB = criarPonto(6.0, 8.0);  
35  
36     printf("Coordenadas do ponto A: ");  
37     imprimirPonto(pA);  
38  
39     printf("Coordenadas do ponto B: ");  
40     imprimirPonto(pB);  
41  
42     Ponto S = somarPontos(pA, pB);  
43  
44     printf("\nA + B = ");  
45     imprimirPonto(S);  
46  
47     return 0;  
48 }
```

TAD Ponto (com ponteiro)

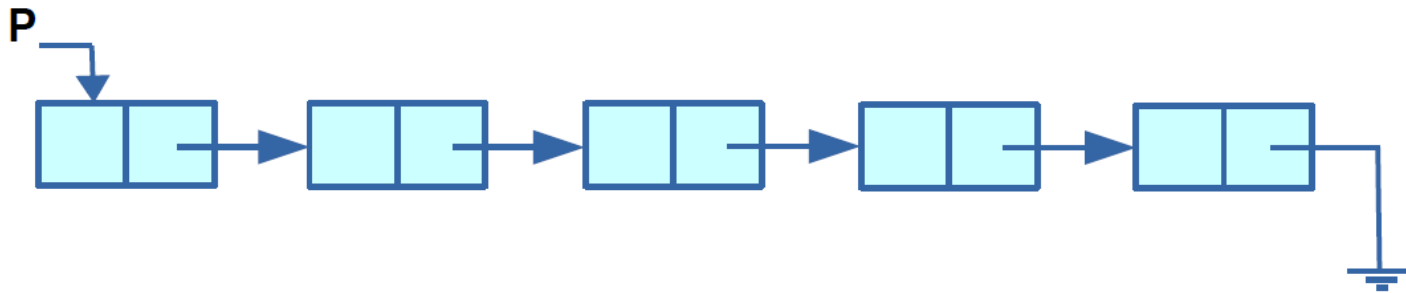
```
Coordenadas do ponto A: (3.00, 4.00)  
Coordenadas do ponto B: (6.00, 8.00)  
  
A + B = (9.00, 12.00)
```

Lista linear ligada/encadeada

- Em uma lista linear sequencial (**estática**) as operações de **inserção** e **remoção** são consideradas **custosas**, pois potencialmente precisamos deslocar vários elementos.
- Para evitar o deslocamento de elementos durante a inserção e remoção utilizaremos uma lista **ligada/encadeada** (**dinâmica**).
- Assim como a lista sequencial a lista ligada/encadeada também é uma **lista linear** (cada elemento possui no máximo um predecessor e um sucessor).

Lista linear ligada/encadeada

É uma **lista linear** na qual **ordem lógica** dos elementos (a ordem “vista” pelo usuário) **não** é a mesma **ordem física** (em memória principal) dos elementos. Desta forma, cada elemento deverá **indicar** quem é seu **sucessor**.

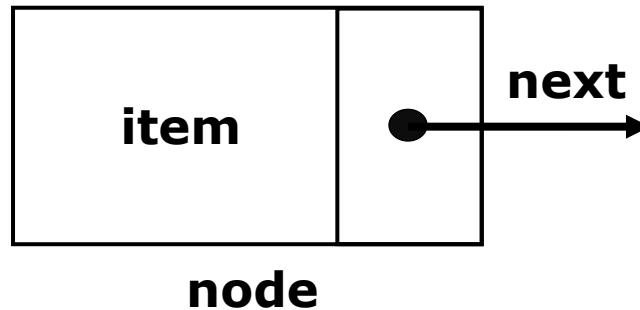


Lista simplesmente ligada/encadeada (alocação dinâmica)

Principais operações em listas

- **Inserção**
 - operação modificadora
- **Busca**
 - operação de consulta (não modificadora)
- **Remoção**
 - operação modificadora

Estrutura para o nó (node) da lista



```
struct node {  
    int item;  
    struct node *next;  
};
```

Endereço de uma lista

- O **endereço** de uma lista ligada/encadeada é o endereço da primeiro nó (node) da lista.
- A lista é vazia (não tem nenhum nó) se o endereço da lista é igual a **NULL**.

Inserção de um elemento na lista

- Em uma lista ligada podemos inserir um elemento em **qualquer posição**, mas vamos fazer isso inicialmente no início. E como temos uma lista que é dinâmica precisamos **alocar memória** para um novo nó.
- A complexidade de tempo da inserção, neste caso, é $O(1)$, ou seja, não depende do tamanho da lista.

Inserção de um elemento na lista

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  struct node {
6      int item;
7      struct node *next;
8  };
9
10 void inserir(struct node **lista, int novoItem) {
11     struct node *novoNode = (struct node *)malloc(sizeof(struct node));
12     novoNode->item = novoItem; // Atribui o valor ao item do nó criado
13     novoNode->next = *lista; // Aponta para o antigo primeiro nó
14     *lista = novoNode; // Atualiza o ponteiro da lista para o novo nó
15 }
16
```

Inserção de um elemento na lista

```
17 void imprimir(struct node *lista) {  
18     struct node *p = lista; // Ponteiro auxiliar para percorrer a lista  
19     printf("Lista: ");  
20     while (p != NULL) { // Itera sobre todos os nós da lista  
21         if (p->next != NULL)  
22             printf("%d -> ", p->item); // Exibe o item seguido de "->"  
23         else  
24             printf("%d", p->item); // Exibe o último item sem "->"  
25         p = p->next; // Avança para o próximo nó  
26     }  
27 }  
28
```

Inserção de um elemento na lista

```
29 int main(void) {
30     int i, n;
31     struct node *Lista = NULL; // Inicializar a lista vazia
32
33     printf("Quantidade de nós da lista: ");
34     scanf("%d", &n);
35     printf("\n");
36
37     // Inserir n elementos (pseudo)aleatórios na lista
38     srand((unsigned)time(NULL));
39
40     for(i = 0; i < n; i++)
41         inserir(&Lista, 1 + (rand()%100));
42
43     // Exibir os elementos da lista
44     imprimir(Lista);
45
46     return 0;
47 }
```




Inserção de um elemento na lista

```
Quantidade de nós da lista: 5  
Lista: 8 -> 18 -> 65 -> 100 -> 45
```

Busca de um elemento na lista

- A busca deve percorrer a lista até encontrar o elemento buscado ou chegar ao final da lista.
- A complexidade de tempo da busca, neste caso, é $O(n)$, sendo n o comprimento da lista.

Busca de um elemento na lista

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  struct node {};
6
7
8
9
10 void inserir(struct node **lista, int novoItem) {}
11
12
13
14
15
16
17 struct node *buscar(struct node *lista, int chave) {
18     struct node *p = lista; // Ponteiro auxiliar para percorrer a lista
19
20     // Percorre a lista enquanto não chegar ao final e não encontrar o elemento
21     while (p != NULL && p->item != chave)
22         p = p->next; // Avança para o próximo nó
23
24     return p; // Retorna o ponteiro para o nó encontrado ou NULL se não achou
25 }
26
27 void imprimir(struct node *lista) {}
28
```

Busca de um elemento na lista

```
39 int main(void) {
40     int i, n;
41     struct node *Lista = NULL; // Inicializar a lista vazia
42     int chave; // Variável para armazenar o valor a ser buscado
43
44     printf("Quantidade de nós da lista: ");
45     scanf("%d", &n);
46     printf("\n");
47
48     // Inserir n elementos (pseudo)aleatórios na lista
49     srand((unsigned)time(NULL));
50
51     for (i = 0; i < n; i++)
52         inserir(&Lista, 1 + (rand()%100));
53
54     // Exibir os elementos da lista
55     imprimir(Lista);
56
57     // Solicitar ao usuário um valor para buscar na lista
58     printf("\n\nDigite o valor do elemnto que deseja buscar: ");
59     scanf("%d", &chave);
60
61     // Buscar o elemento na lista ligada
62     if (buscar(Lista, chave) != NULL)
63         printf("\nO elemento %d foi encontrado na lista!", chave);
64     else
65         printf("\nO elemento %d não foi encontrado na lista!", chave);
66
67     return 0;
68 }
```

Busca de um elemento na lista




```
Quantidade de nós da lista: 5  
  
Lista: 98 -> 65 -> 43 -> 74 -> 31  
  
Digite o valor do elemnto que deseja buscar: 74  
  
O elemento 74 foi encontrado na lista!
```

```
Quantidade de nós da lista: 5  
  
Lista: 22 -> 18 -> 27 -> 97 -> 63  
  
Digite o valor do elemnto que deseja buscar: 30  
  
O elemento 30 não foi encontrado na lista!
```


Remoção de um elemento da lista

- A remoção deve percorrer a lista até encontrar o elemento que se deseja remover ou chegar ao final da lista.
- A complexidade de tempo da remoção, neste caso, é $O(n)$, sendo n o comprimento da lista.

Remoção de um elemento da lista

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  struct node {};
9
10 void inserir(struct node **lista, int novoItem) {}
16
17 struct node *buscar(struct node *lista, int chave) {}
26
```

Remoção de um elemento da lista

```
27 struct node *remove(struct node *lista, int chave) {
28     struct node *p = lista; // Ponteiro p inicia no primeiro nó da lista
29     struct node *q = NULL; // Ponteiro q para rastrear o nó anterior de p
30
31     // Percorre a lista procurando o nó com o valor chave
32     while (p != NULL && p->item != chave) {
33         q = p; // Armazena o nó atual p em q (nó anterior ao nó p)
34         p = p->next; // Avança para o próximo nó
35     }
36
37     // Se o elemento não foi encontrado, a lista permanece inalterada
38     if (p == NULL)
39         return lista; // Retorna a lista original, sem remoção
40
41     // Se o nó a ser removido for o primeiro da lista
42     if (q == NULL)
43         lista = p->next; // O próximo nó de p torna-se o novo início da lista
44     else
45         q->next = p->next; // Ajusta o ponteiro next do nó q para p->next
46
47     free(p); // Libera a memória do nó removido
48     return lista;
49 }
50
```

Remoção de um elemento da lista

```
51 void imprimir(struct node *lista) {  
52  
53 int main(void) {  
54     int i, n;  
55     struct node *Lista = NULL; // Inicializar a lista vazia  
56     int chave; // Variável para armazenar o valor a ser removido  
57  
58     printf("Quantidade de nós da lista: ");  
59     scanf("%d", &n);  
60     printf("\n");  
61  
62     // Inserir n elementos (pseudo)aleatórios na lista  
63     srand((unsigned)time(NULL));  
64  
65     for (i = 0; i < n; i++)  
66     {  
67         inserir(&Lista, 1 + (rand()%100));  
68     }  
69  
70     // Exibir os elementos da lista antes da remoção  
71     printf("Lista antes da remoção:\n");  
72     imprimir(Lista);  
73  
74     // Solicitar ao usuário um valor para remover na lista  
75     printf("\n\nDigite o valor do elemento que deseja remover: ");  
76     scanf("%d", &chave);  
77  
78     Lista = remover(Lista, chave);  
79  
80     // Exibir os elementos da lista depois da remoção  
81     printf("\nLista depois da remoção:\n");  
82     imprimir(Lista);  
83  
84     return 0;  
85 }  
86  
87  
88  
89  
90  
91  
92  
93 }
```

Remoção de um elemento da lista

```
Quantidade de nós da lista: 5
```

```
Lista antes da remoção:
```

```
Lista: 21 -> 62 -> 1 -> 100 -> 60
```

```
Digite o valor do elemento que deseja remover: 1
```

```
Lista depois da remoção:
```

```
Lista: 21 -> 62 -> 100 -> 60
```

```
Quantidade de nós da lista: 5
```

```
Lista antes da remoção:
```

```
Lista: 67 -> 83 -> 96 -> 29 -> 16
```

```
Digite o valor do elemento que deseja remover: 50
```

```
Lista depois da remoção:
```

```
Lista: 67 -> 83 -> 96 -> 29 -> 16
```

Implementação de uma lista com cabeça

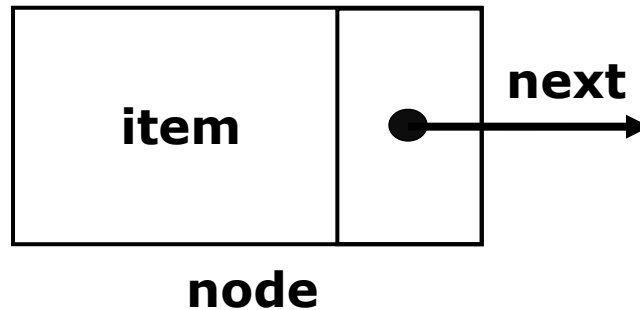
Podemos implementar uma lista ligada/encadeada de 2 maneiras diferentes.

- “**Lista sem cabeça**”: O conteúdo do primeiro nó é tão relevante quanto o dos demais nós da lista.
- “**Lista com cabeça**”: O primeiro nó da lista serve apenas para indicar o início da lista (seu conteúdo é considerado “irrelevante”).

Implementação de uma lista com cabeça

- Em alguns casos o nó cabeça (*head node* ou *dummy node*) pode conter informações úteis como, por exemplo, o número de elementos da lista ou se a lista está ordenada ou não.
- Listas com cabeça não reduzem a complexidade assintótica, mas simplificam a implementação dos algoritmos.

Estrutura para o nó (node) da lista



```
typedef struct node *link;  
  
struct node {  
    int item;  
    link next;  
};
```


Implementação de uma lista com cabeça

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  // Definição do tipo link como ponteiro para struct node
6  typedef struct node *link;
7
8  // Definição da estrutura node
9  struct node {
10     int item;
11     link next; // Ponteiro para o próximo nó
12 };
13
```

Implementação de uma lista com cabeça

```
14 // Função para inserir um item na lista com cabeça
15 void inserir(link *lista, int novoItem) {
16     link novoNode = (link)malloc(sizeof(struct node));
17     novoNode->item = novoItem; // Atribui o valor ao item do nó criado
18     novoNode->next = (*lista)->next; // Aponta para o antigo primeiro nó
19     (*lista)->next = novoNode; // Atualiza o ponteiro da lista para o novo nó
20 }
21
```

Implementação de uma lista com cabeça

```
22 // Função para buscar um item na lista com cabeça
23 struct node *buscar(link lista, int chave) {
24     link p = lista->next; // Ponteiro auxiliar começa no primeiro nó real (depois da cabeça)
25
26     // Percorre a lista enquanto não chegar ao final e não encontrar o elemento
27     while (p != NULL && p->item != chave)
28         p = p->next; // Avança para o próximo nó
29
30     return p; // Retorna o ponteiro para o nó encontrado ou NULL se não achou
31 }
32
```

Implementação de uma lista com cabeça

```
33 // Função para remover um item na lista com cabeça
34 struct node *remove(link lista, int chave) {
35     link p = lista->next; // Ponteiro p inicia no primeiro nó real (depois da cabeça)
36     link q = lista; // Ponteiro q é o nó anterior (cabeça) de p
37
38     // Percorre a lista procurando o nó com o valor chave
39     while (p != NULL && p->item != chave) {
40         q = p; // Armazena o nó atual p em q (nó anterior ao nó p)
41         p = p->next; // Avança para o próximo nó
42     }
43
44     // Se o elemento não foi encontrado, a lista permanece inalterada
45     if (p == NULL)
46         return lista; // Retorna a lista original, sem remoção
47
48     // Remove p da lista, ajustando o ponteiro do nó anterior
49     q->next = p->next;
50
51     free(p); // Libera a memória do nó removido
52     return lista;
53 }
54
```

Implementação de uma lista com cabeça

```
55 // Função para imprimir a lista com cabeça
56 void imprimir(link lista) {
57     link p = lista->next; // Começa no primeiro nó real (depois da cabeça)
58     printf("Lista: ");
59     while (p != NULL) { // Itera sobre todos os nós da lista
60         if (p->next != NULL)
61             printf("%d -> ", p->item); // Exibe o item seguido de "->"
62         else
63             printf("%d", p->item); // Exibe o último item sem "->"
64         p = p->next; // Avança para o próximo nó
65     }
66 }
67
```

Implementação de uma lista com cabeça

```
68 int main(void) {
69     int i, n;
70     link Lista = (link)malloc(sizeof(struct node)); // Cria a cabeça da lista
71     Lista->next = NULL; // Inicializar a lista com cabeça vazia
72     int chave; // Variável para armazenar o valor a ser buscado/removido
73
74     printf("Quantidade de nós da lista: ");
75     scanf("%d", &n);
76     printf("\n");
77
78     // Inserir n elementos (pseudo)aleatórios na lista
79     srand((unsigned)time(NULL));
80
81     for (i = 0; i < n; i++)
82         inserir(&Lista, 1 + (rand()%100));
83
84     // Exibir os elementos da lista
85     imprimir(Lista);
86 }
```

Implementação de uma lista com cabeça

```
87 // Solicitar ao usuário um valor para buscar na lista
88 printf("\n\nDigite o valor do elemento que deseja buscar: ");
89 scanf("%d", &chave);
90
91 // Buscar o elemento na lista ligada
92 if (buscar(Lista, chave) != NULL)
93     printf("\nO elemento %d foi encontrado na lista!", chave);
94 else
95     printf("\nO elemento %d não foi encontrado na lista!", chave);
96
97 // Solicitar ao usuário um valor para remover na lista
98 printf("\n\nDigite o valor do elemento que deseja remover: ");
99 scanf("%d", &chave);
100
101 Lista = remover(Lista, chave);
102
103 // Exibir os elementos da lista depois da remoção
104 printf("\nLista depois da remoção:\n");
105 imprimir(Lista);
106
107 return 0;
108 }
```

Implementação de uma lista com cabeça

```
Digite o valor do elemento que deseja buscar: 94  
O elemento 94 foi encontrado na lista!  
Digite o valor do elemento que deseja remover: 94  
Lista depois da remoção:  
Lista: 74 -> 95 -> 45 -> 14
```


Fim!