



Lista Sequencial

Aula 02

Marcos Silvano Almeida
marcossilvano@utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

TAD _ Tipos Abstratos de Dados



- Um tipo abstrato é aquele que não define a forma como será implementado, mas sim, o modelo que representa e as operações possíveis
 - O TAD não define a estratégia de implementação
 - Na implementação, vamos usar estruturas de dados, como arrays e structs, para construir o TAD.
- Vamos conhecer os seguintes TADs
 - Lista Sequencial
 - Lista Encadeada
 - Lista Duplamente Encadeada
 - Lista Circular
 - Fila
 - Pilha

Lista Sequencial

- Sequência de dados armazenados contiguamente.
 - Permite acesso **sequencial** e **aleatório** aos seus elementos
 - Elemento são acessados por índice (int)
 - Operações típicas:
 - Criar e Destruir
 - Adicionar, Inserir e Remover
 - Acessar e Pesquisar



Lista Sequencial



- Características
 - Acesso **sequencial** e **aleatório**.
 - **Bom** para para inserções/remoções ao **final da lista**
 - **Ruim** para inserções/remoções no **início ou meio da lista**.

```
lista ← create()
```

lista →

size: 0

```
lista.add(5)
```

```
lista.add(9)
```

```
lista.add(15)
```

```
lista.add(4)
```

lista →

5	9	15	4
0	1	2	3

size: 4

lista →

5		9	15	4
0	1	2	3	4

size: 4

```
lista.insert(1, 7)
```

lista →

5	7	9	15	4
0	1	2	3	4

size: 5

Lista Sequencial

- Python não possui a estrutura de dados Array, mas o TAD Lista

<code>lista = []</code>	<code>lista -> []</code>	<code>size: 0</code>
<code>lista.append(5)</code>	<code>lista -> [5]</code>	<code>size: 1</code>
<code>lista.append(9)</code>	<code>lista -> [5, 9]</code>	<code>size: 2</code>
<code>lista.append(15)</code>	<code>lista -> [5, 9, 15]</code>	<code>size: 3</code>
<code>lista.append(4)</code>	<code>lista -> [5, 9, 15, 4]</code>	<code>size: 4</code>
<code>lista.insert(1, 7)</code>	<code>lista -> [5, 7, 9, 15, 9]</code>	<code>size: 5</code>

`lista ← create()` `lista →` `size: 0`

`lista.add(5)`
`lista.add(9)`
`lista.add(15)`
`lista.add(4)`

`lista →`

5	9	15	4
0	1	2	3

`size: 4`

`lista.insert(1, 7)` `lista →` `size: 5`

5	7	9	15	4
0	1	2	3	4



Implementação da Lista Sequencial

Lista Sequencial

- Implementada clássica utiliza vetor
 - Vetor redimensionável, alocado em Heap
 - TAD será definida como um struct, contendo:
 - Capacidade do vetor → tamanho real do vetor
 - Tamanho da lista → quantidade de elementos na lista (posições ocupadas)
 - Ponteiro para o vetor alocado em heap

```
struct VectorList {  
    int size;           // número de posições ocupadas  
    int capacity;       // capacidade da lista (tamanho do vetor)  
    int* data;          // endereço do vetor na heap  
};
```



O tipo Lista Sequencial

- Manipulação da Lista Sequencial ocorre por meio das operações (funções)

```
VectorList* list1 = list_create(10);  
list_push_back(list1, 5); // adiciona ao final  
list_push_back(list1, 9);  
list_push_back(list1, 15);  
list_push_back(list1, 5);  
list_push_back(list1, 7);  
list_pop_back(list1);      // remove do final  
list_print(list1);
```



Operações de criação e acesso



```
// Cria e prepara lista. Retorna NULL se erro.
```

```
VectorList* list_create(int capacity);
```

```
// Libera memoria alocada para o vetor.
```

```
VectorList* list_destroy(VectorList* this);
```

```
// Realoca vetor interno e move elementos. Retorna NULL se erro.
```

```
bool list_resize(VectorList* this);
```

```
void list_print(VectorList* this); // Imprime o conteúdo da lista.
```

```
int list_get(VectorList* this, int pos); // Obtém elemento na posição
```

```
int list_find(VectorList* this, int elem); // Devolve posição do elemento ou -1
```

```
bool list_is_empty(VectorList* this); // Informa se lista vazia.
```

```
bool list_is_full(VectorList* this); // Informa se lista cheia.
```

Operações de modificação



```
// Insere elemento ao final da lista (false/erro)
```

```
bool list_push_back(VectorList* this, int elem);
```

```
// Remove elemento do final da lista (false/erro)
```

```
bool list_remove(VectorList* this);
```

```
// Insere elemento na posição especificada, abrindo espaço para o novo elemento
```

```
bool list_insert(VectorList* this, int elem, int pos);
```

```
// Insere elemento em ordem crescente
```

```
bool list_insert_sorted(VectorList* list, int elem);
```

```
// Remove elemento da posição especificada e ajusta lista (fecha “buraco”)
```

```
bool list_remove(VectorList* this, int pos);
```