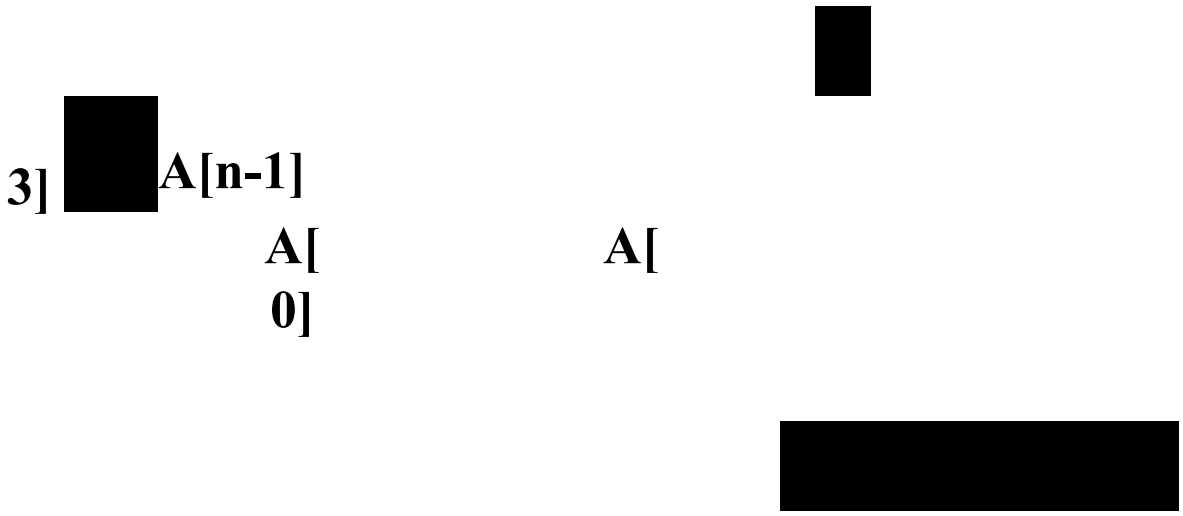
Estruturas estáticas: podem armazenar até uma quantidade fixa de elementos, que deve ser indicada quando ela é criada;

- ***Estruturas dinâmicas***: o tamanho e a capacidade variam de acordo com a demanda, a medida que o programa vai sendo executado. Em geral, são construídas com ponteiros/referências.

Estruturas que armazenam uma quantidade fixa de elementos do mesmo tipo. O acesso a um elemento é feito a partir do índice do elemento desejado.



Arrays não podem armazenar mais elementos do que o seu tamanho, logo, o tamanho deve ser o máximo necessário.

Quando a quantidade de elementos é variável, o uso de arrays pode desperdiçar memória, já que nem todas as suas posições são necessariamente ocupadas.

Espaço utilizado

Espaço desperdiçado

Estruturas criadas para evitar o desperdício de memória, alocando apenas o espaço necessário para seus dados.

A construção é feita a partir de ponteiros/referências.

5 8 1 4

Antes da inserção:

Após a

Listas

Encadeadas

Ao contrário de um array, uma lista não pode acessar seus elementos de modo direto, e sim, de modo sequencial, ou seja, um por vez

A estrutura **Lista** geralmente contém uma referência para o primeiro elemento da lista (*NoLista inicio*), a partir do qual todos os outros poderão ser acessados.

Armazenam uma quantidade variável de elementos do mesmo tipo

Lista de inteiros

Lista de referências para objetos da classe String

“Algoritmos” “Estruturas” “Dados”

Listas são formadas por estruturas chamadas **nós**. Um nó é uma estrutura **auto-referencial**, isto é, contém uma referência para outra estrutura do mesmo tipo

Dado armazenado no nó atual

Referência para o próximo elemento da lista

Indicador do fim da lista (referência **null**)

Ex: Nó de uma lista de inteiros

```
class NoLista {
 int valor;
NoLista next  }
```

Elemento do nó

Referência para o nó seguinte

Ex: Nó de uma lista de objetos da classe String

```
class NoLista {

 String nome;
NoLista next; }
```

Elemento do nó

Referência para o nó seguinte

O fim de uma lista é indicada por uma referência nula (representada por um **X**), ou seja, o último nó de uma lista referencia como o próximo elemento o **null**

valor / next valor / next

```
public NoLista (int valor) {
 this.valor = valor;
this.next = null; }
```

(referência **null**)

Indicador do fim da lista

Para criar a lista propriamente dita, criaremos a classe **Lista**, que manipula objetos do tipo **NoLista**

```
class Lista {
 NoLista inicio;

 public Lista() {
 this.inicio = null;
 }
 // insere valor no começo da lista
 public void inserir(int valor) {...}
 // insere valor no fim da lista
 public void inserirNoFim(int valor) {...}
}
```

Para inserir um elemento em um array, pode ser necessário expandi-lo e copiar os elementos um a um:

9 13 2 10 3 7 2 8

9 13 2 10 3 7 2 8 5

Em uma lista, basta criar um nó, encontrar a posição desejada e inseri-lo

Inserção em

Listas

Toda operação (inserção, remoção, busca etc.) é feita a partir de um NoLista armazenado na classe Lista. Se o NoLista não existir (referência nula), a lista está vazia

```
if (lista.inicio == null) {
  ...
}
```

Se estiver vazia, na inserção, atribua o início ao novo nó. Caso contrário, encontre o local correto para inserir o novo elemento.

Se a lista estiver vazia:

novo nó

Caso contrário, inserindo no fim da lista teremos:

3 1 2

9

último nó?
NÃO

último nó?
NÃO

último nó?
SIM

novo nó

Inserção em

Listas

```
public void inserirNoFim(int valor) {
if (this.inicio == null) {   // lista
vazia
 this.inicio = new NoLista(valor);   }
else {
 // procura pelo fim da lista
 NoLista atual = this.inicio;
while (atual.next != null)
atual = atual.next;
 // insere o nó no fim da lista
 atual.next = new NoLista(valor);   }
 }
```

Inserção em

Listas

```
public void inserir(int valor) {
if (this.inicio == null) {
  // lista vazia, então só é preciso criar o nó
this.inicio = new NoLista(valor);  }
else {

  // cria-se novo no e atualiza o NoLista inicio
NoLista novoNo = new NoLista(valor);

 novoNo.next = this.inicio;

 this.inicio = novoNo;
 }
}
```

Para inserir um elemento em um array em uma posição qualquer, pode ser necessário mover vários elementos:

2 3 4 7 8 9 10 13 2 3 4 

5 7 8 9 10 13

Da mesma maneira, em uma lista, basta criar um nó, encontrar a posição desejada e inseri-lo

2 3 4 7 8 9 10 13 ~~5~~

Para inserir um nó entre dois outros nós:

anterior anterior.next

5

novo nó
anterior.next

```
NoLista novoNo = new NoLista(5);

novoNo.next = anterior.next;
anterior.next = novoNo;
```

Para remover um elemento de uma posição qualquer do array, pode ser necessário mover vários elementos:

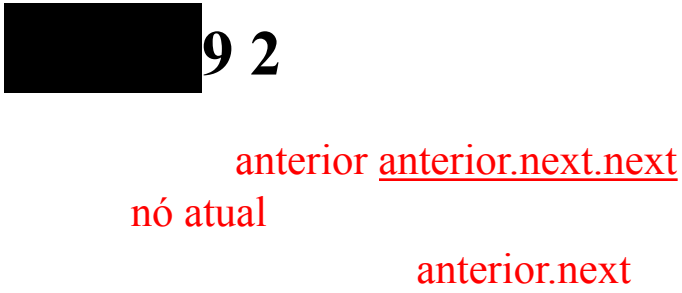
2 3 4 7 8 9 10 13 5

2 3 4 7 8 9 10 13

Para remover um elemento de uma lista, basta encontrar o nó correspondente e alterar os ponteiros

2 3 4 5 7 8 9 10 13

Para excluir um nó entre dois outros nós:




```
anterior.next = anterior.next.next;
```

• Listas podem apresentar diversas funções de inserção/remoção (no começo, no fim, em qualquer parte da lista).

•Pilhas seguem o padrão LIFO(Last-In-First-Out), usando apenas as funções push(inserir no começo) e pop(remover do começo)

•Filas seguem o padrão FIFO(First-In-First-Out), usando apenas as funções queue(inserir no fim) e dequeue (remover do começo)

•Sendo assim, implementar pilhas e filas a partir de listas é simples, já que listas incluem as funções do padrão LIFO e FIFO.

• Arrays podem ocupar espaço desnecessário na memória, mas seu acesso é feito diretamente

• Listas ocupam apenas o espaço necessário, mas é preciso espaço extra para armazenar as referências. Além disso, seu acesso é seqüencial, ou seja, a busca inicia por um nó, depois vai pra outro nó, e assim por diante, até que se encontre o nó procurado.

•Listas duplamente encadeadas (dois ponteiros dentro de cada nó, um para o próximo nó e outro pro anterior) dão maior poder de implementação das funções, apesar dos custos adicionais de memória por conta do número de ponteiros.