

Algoritmos com **vetores**

Geração, aleatorização, buscas e deslocamentos

Prof. Manuel F. Paradela Ledón



Vetores (arrays, arranjos unidimensionais)

Vetor de n elementos

a_1 a_2 a_3 ... a_n

em Álgebra Linear

ou

a_0 a_1 a_2 ... a_{n-1}

em linguagens de programação

Exemplo: um vetor de $n = 4$ elementos

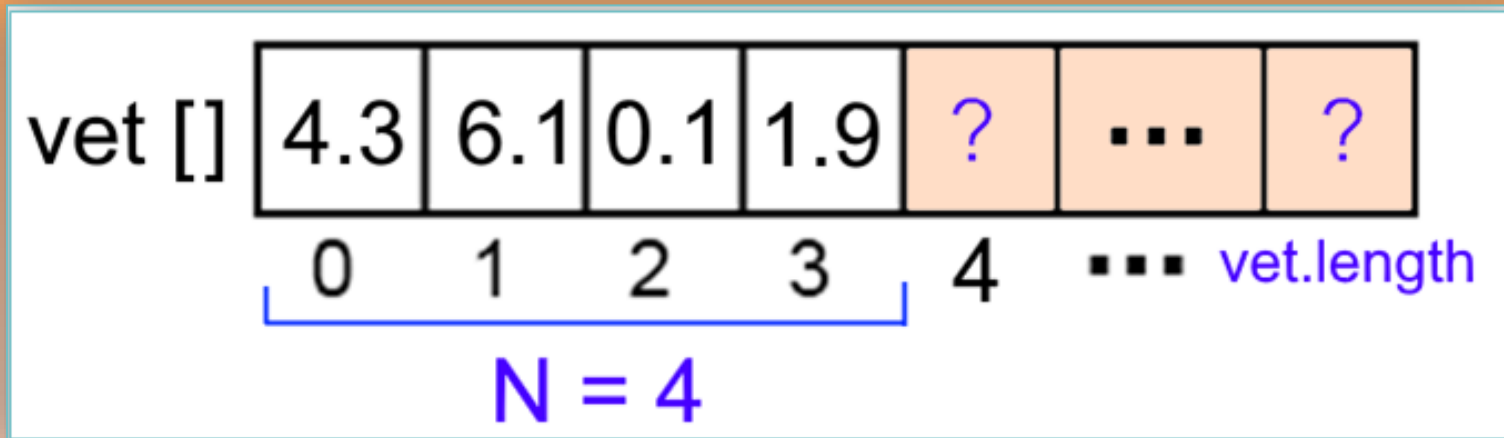
ítems:

| $a[0]$ | $a[1]$ | $a[2]$ | $a[3]$ |
|--------|--------|--------|--------|
| 12.3 | 4.1 | -12.7 | 6.5 |

posições:

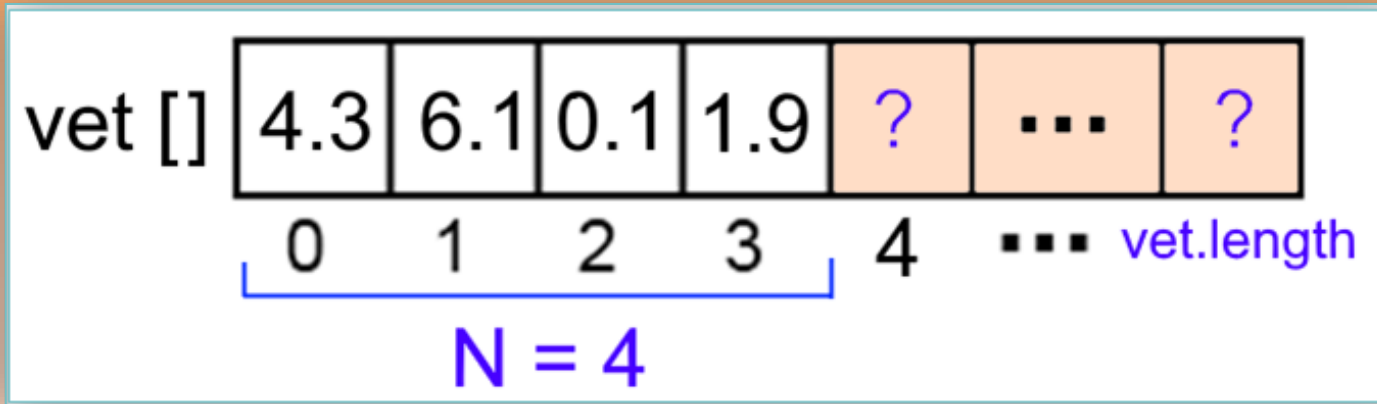
0 1 2 3

Vetores (arrays, arranjos unidimensionais)



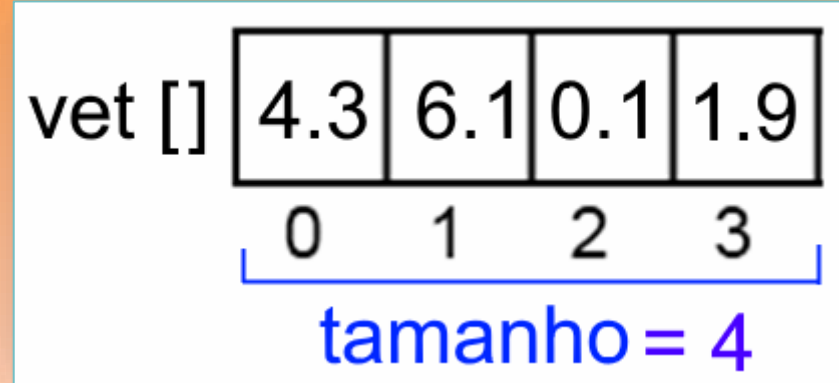
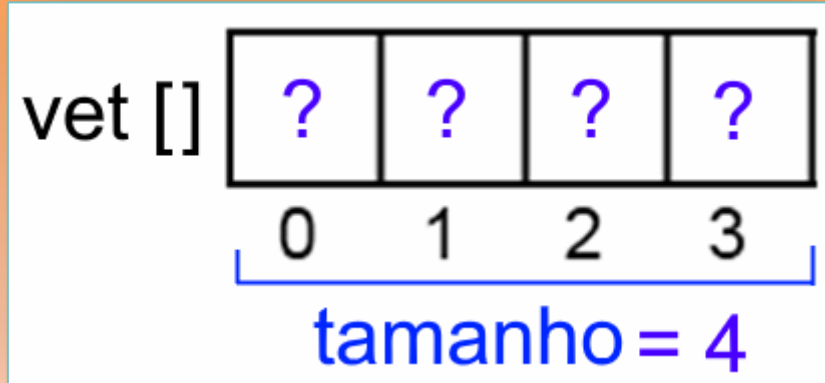
- Podemos alocar memória para o vetor e utilizar somente uma parte do espaço alocado.
- Observe na figura acima que N (N=4 elementos) é menor que o tamanho real/máximo do vetor (vet.length) e que as posições com ? não são utilizadas.
- Normalmente, a primeira posição do vetor é 0.
- Algumas linguagens de programação poderiam permitir redimensionar um vetor. Em Java, por exemplo, para efetuar o redimensionamento será necessário criar um novo vetor e efetuar a cópia dos itens/elementos do vetor original para o novo vetor.

Vetores (arrays, arranjos unidimensionais)



- Para acessar um elemento utilizamos a notação **`vet[i]`**, sendo `vet` o nome do vetor e `i` a posição que queremos referenciar. Por exemplo, com o comando **`double v = vet[3];`** guardaremos o valor 1.9 na variável `v`.
- Também, para alterar o valor de um elemento do vetor, por exemplo, para guardar o valor 5.2 na posição 2, podemos executar o comando **`vet[2] = 5.2;`** que irá substituir o valor 0.1.

Vetores (arrays, arranjos unidimensionais)



- Ao alocar memória para um vetor, os valores dos itens/elementos estão inicialmente indefinidos (1ª figura).
- Posteriormente, utilizaremos algum algoritmo para armazenar valores específicos nas diferentes posições do vetor (2ª figura).
- Nos próximos slides apresentamos algoritmos (exemplos na linguagem de programação Java) para criar e guardar valores aleatórios nas diferentes posições de um vetor.




Exercícios para fazer no laboratório

Utilizando como base o projeto em **ExerciciosLaboratorio.zip**, adicione e teste estes dois métodos:

1. **calcularSomaDosElementos(double vet [])**, que calcule e retorne a soma de todos os elementos de um vetor de elementos reais qualquer vet.
2. **calcularSomaDosElementosPositivos(double vet [])**, que calcule e retorne a soma dos elementos positivos (maiores que zero) de um vetor de elementos reais qualquer vet.

Analise também estes três exemplos com vetores (arquivos .zip com projetos NetBeans dentro)



| | |
|---|-------------------------------|
|  | LeituraDeUmVetor |
|  | LeituraDeUmVetorComScanner |
|  | LeituraDeUmVetorComScannerv02 |

Atribuir valores aleatórios
(quaisquer) a um vetor

Exemplo: colocando valores aleatórios em um vetor

```
public Ex_random() {  
    //Geração aleatória dos itens do vetor:  
    float vet[] = new float[10]; //alocamos memória para o vetor vet  
    for (int i = 0; i < vet.length; i++) {  
        //gera um valor aleatório e o guarda na posição i do vetor vet:  
        vet[i] = geraFloat();  
    }  
    System.out.println("\nVetor de 10 elementos gerados aleatoriamente");  
    System.out.println("com valores entre 0 e 9,999: \n");  
    visualizaVetor(vet);  
}
```


```
public float geraFloat() {  
    //Oracle, nextFloat(): "Retorna o próximo valor real pseudo-aleatório  
    //uniformemente distribuído entre 0,0 e 1,0 a partir da sequência  
    //deste gerador de números aleatórios." intervalo: de 0.0 (incluso) a 1.0 (não inclusivo)  
    Random rnd = new Random();  
    float numero = rnd.nextFloat();  
    return (numero * 10);  
}
```

a classe Random tem as funções: nextFloat,
nextInt, nextDouble, nextLong, nextBoolean...

Exemplo no projeto Ex_random

Reposicionar (embaralhar) os valores já
armazenados em um vetor
(aleatorização)

Algoritmo para 'organizar' (embaralhar) aleatoriamente um vetor (uma lista ArrayList). Algoritmo de Fisher-Yates.

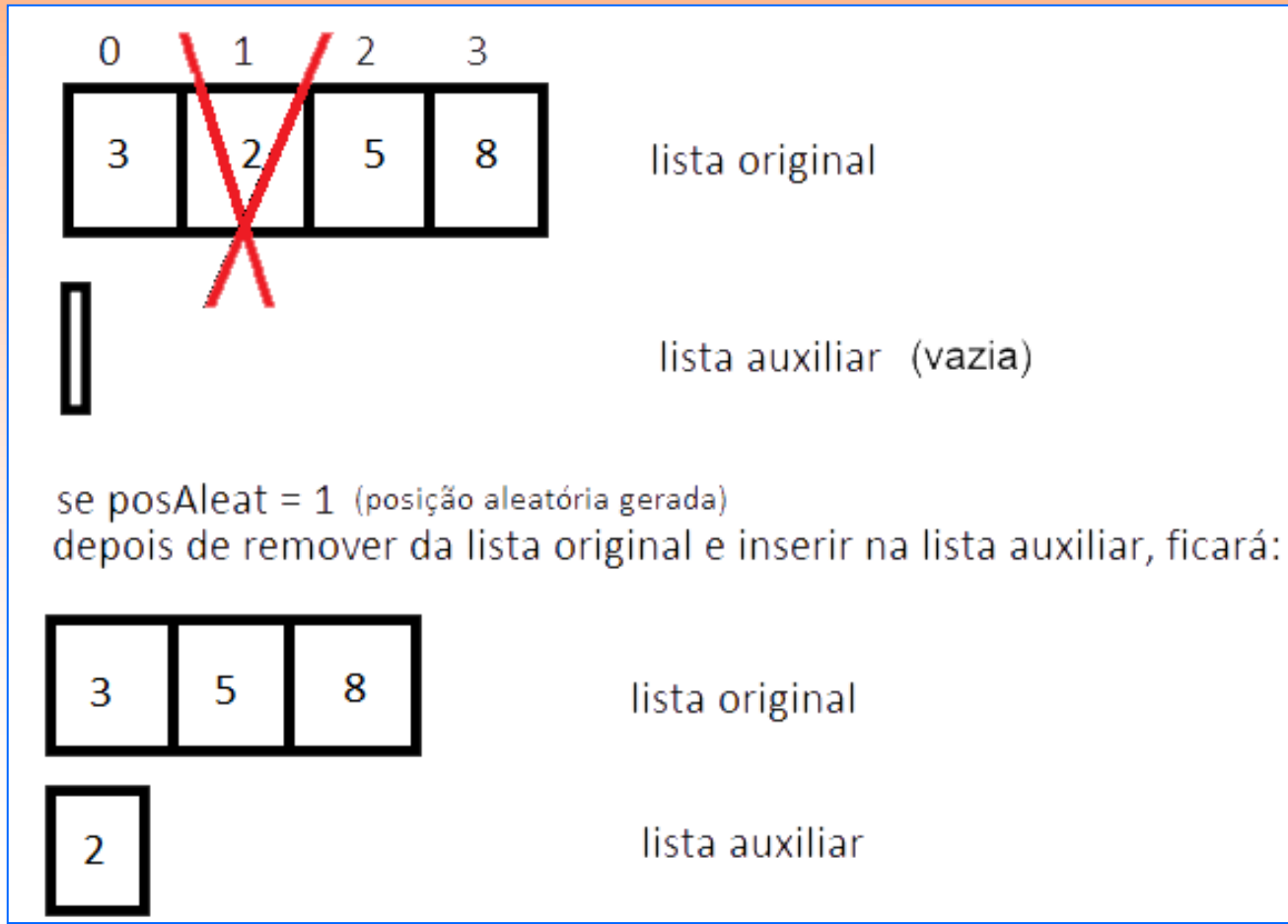
```
ArrayList lista = new ArrayList();
lista.add(1.2f); lista.add(4.3f); lista.add(6.1f); lista.add(-7.7f);
lista.add(0.4f); lista.add(-8.8f); lista.add(9.0f); lista.add(3.3f);
System.out.println("\nLista original:");
visualizaArrayList(lista);
aleatorizar(lista); 
System.out.println("\nLista anterior aleatorizada com o algoritmo de Fisher-Yates:");
visualizaArrayList(lista);
```

```
public void visualizaArrayList(ArrayList lista) {
    for (int i=0; i < lista.size(); i++) {
        System.out.print(lista.get(i) + "  ||  ");
    }
    System.out.println();
}
```

Exemplo no projeto Ex_random

Algoritmo de Fisher-Yates (aleatorização de um vetor)

O algoritmo de Fisher-Yates utiliza uma **lista auxiliar**, onde serão inseridos os elementos que foram selecionados aleatoriamente da **lista original**. O elemento sorteado na lista original será eliminado. Na figura a seguir demonstramos a lógica que será executada se for sorteada a posição 1 da lista original, em um exemplo com uma lista inicialmente com quatro elementos 3, 2, 5, 8.



Algoritmo de Fisher-Yates (aleatorização de um vetor)

```
public void aleatorizar(ArrayList lista) { //parâmetro por referência
    // Algoritmo de Fisher-Yates, implementado em Java com ArrayList
    ArrayList listaTemp = new ArrayList(); // criamos uma lista auxiliar adicional
    Random rnd = new Random();
    while ( lista.size() > 1 ) {
        // selecionamos aleatoriamente uma posição da lista inicial:
        int posAleat = rnd.nextInt(lista.size());
        // adicionamos o elemento sorteado no final da lista auxiliar:
        listaTemp.add(lista.get(posAleat));
        // eliminamos o elemento da lista original:
        lista.remove(posAleat);
    }
    // adicionamos o único restante no final da lista adicional:
    listaTemp.add(lista.get(0));
    lista.clear(); //limpamos a lista original
    // por último, passamos todos os elementos da lista adicional para a lista original
    for (int i=0; i < listaTemp.size(); i++) {
        lista.add(listaTemp.get(i));
    }
}
```

Exemplo no projeto Ex_random

nextInt(int bound)

Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Busca sequencial em um vetor

Busca sequencial (linear) em um vetor

| | | | | | | | |
|--------|-----|-----|-----|-----|-----|-----|-----|
| vet [] | 4.3 | 6.1 | 0.1 | 1.9 | 2.4 | 7.2 | 5.5 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Buscar: 2.4

- O valor a buscar poderá ser encontrado ou não.
- Como o vetor poderá estar desordenado, a busca começará sempre na posição 0 (zero) e terminará se o valor foi encontrado ou se ultrapassamos a última posição do vetor.
- No pior dos casos (quando o valor se encontra na última posição ou não se encontra no vetor) serão necessárias n comparações, considerando que n é o tamanho do vetor. Então, a complexidade deste algoritmo é de $O(n)$. Veja que no melhor dos casos (se o elemento buscado for o primeiro do vetor) temos $O(1)$, no pior caso $O(n)$ e na média temos $O(n/2)$, simplificada como $O(n)$.

Busca sequencial (linear) em um vetor

```
public class Buscas {  
  
    public static void main (String args[]) {  
        new Buscas();  
    }  
  
    public Buscas() {  
        double a [] = {4.3, 6.1, 0.1, 1.9, 2.4, 7.2, 5.5};  
        int pos = buscaSequencial (a, 2.4); // busca o valor 2.4 no vetor a  
        if (pos != -1) System.out.println("O valor foi encontrado na posição " + pos);  
        else System.out.println("O valor não foi encontrado.");  
    }  
  
    int buscaSequencial (double vet[], double buscado) { //busca sequencial num vetor (double)  
        for (int i = 0; i < vet.length; i++) {  
            if (vet[i] == buscado) return i; // encontramos o valor buscado, retornamos i  
        }  
        return -1; // o item não se encontra no vetor  
    }  
  
}
```

Exemplo no projeto Buscas

Busca sequencial (linear) em um trecho de um vetor

```
public Buscas() {  
    //...  
    double a [] = {4.3, 6.1, 0.1, 1.9, 2.4, 7.2, 5.5};  
    int pos = buscaSequencial (a, 2.4, 0, 3); // busca o valor 2.4 no vetor a, entre 0 e 3  
    if (pos != -1) System.out.println("O valor foi encontrado na posição " + pos);  
        else System.out.println("O valor não foi encontrado nesse trecho.");  
}  
  
int buscaSequencial (double vet[], double buscado, int de, int ate) { //busca em vetor real  
    for (int i = de; i <= ate; i++) { //ambas as posições de e ate serão consideradas  
        if (vet[i] == buscado) return i; // encontramos o valor buscado  
    }  
    return -1; // o item não se encontra nesse trecho do vetor  
}
```

Exemplo no projeto Buscas

Busca binária em um vetor ordenado

Busca binária em um vetor ordenado

Buscando o valor **6.1** dentro do vetor vet

| | | | | | | | |
|--------|-----|-----|-----|-----|-----|-----|-----|
| vet [] | 4.3 | 6.1 | 7.1 | 8.9 | 9.0 | 9.7 | 9.8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| | | | | | | | |
|--------|-----|-----|-----|------|-----|-----|-----|
| vet [] | 4.3 | 6.1 | 7.1 | 8.9 | 9.0 | 9.7 | 9.8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | inf | | | meio | | | sup |

| | | | | | | | |
|--------|-----|------|-----|-----|-----|-----|-----|
| vet [] | 4.3 | 6.1 | 7.1 | 8.9 | 9.0 | 9.7 | 9.8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | inf | meio | sup | | | | |

Utilizando um **vetor ordenado** (crescente na figura), selecionamos a posição central (meio) e verificamos se esse é o elemento buscado.

Se não for o item buscado, buscaremos no trecho inferior ou no trecho superior, dependendo da comparação do valor buscado com esse elemento central. O processo se repetirá para cada novo trecho.

A busca terminará se encontramos o elemento ou se os ponteiros inf e sup já se cruzaram.

A complexidade deste método é $O(\log_2 n)$ no pior caso e $O(1)$ no melhor.

Busca binária em um vetor ordenado (algoritmo iterativo)

```
double b [] = {4.3, 6.1, 7.1, 8.9, 9.4, 9.6, 10.5}; // vetor ordenado
pos = buscaBinaria (b, 9.4);
if (pos != -1) System.out.println("O valor foi encontrado na posição " + pos);
    else System.out.println("O valor não foi encontrado no vetor.");
```

```
int buscaBinaria (double vet[], double buscado) {
    int inf = 0; // limite inferior
    int sup = vet.length - 1; // limite superior
    int meio;
    while (inf <= sup) {
        meio = (inf + sup) / 2;
        if ( buscado == vet [meio] ) return meio; // o valor buscado foi encontrado!
        if ( buscado < vet [meio] ) sup = meio - 1; else inf = meio + 1;
    }
    return -1; // o valor buscado não foi encontrado!
    // poderíamos retornar -(inf + 1) para especificar que: não foi encontrado
    // e que a posição de inserção para esse valor buscado seria -inf - 1
}
```

| | | | | | | | |
|--------|-----|-----|-----|------|-----|-----|-----|
| vet [] | 4.3 | 6.1 | 7.1 | 8.9 | 9.0 | 9.7 | 9.8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | inf | | | meio | | | sup |

Exemplo no projeto Buscas

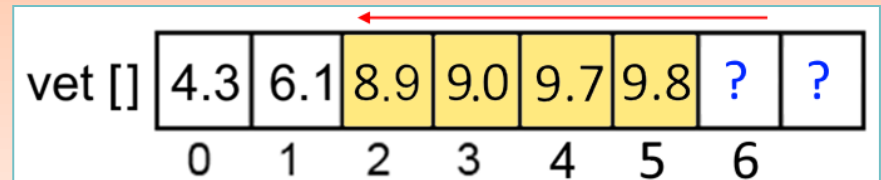
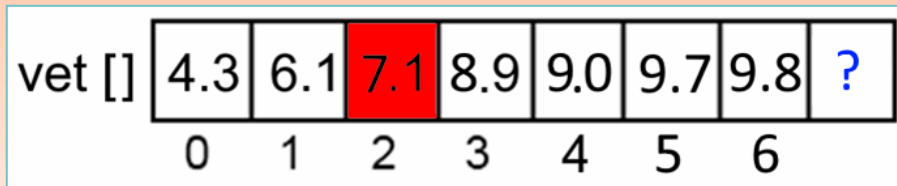
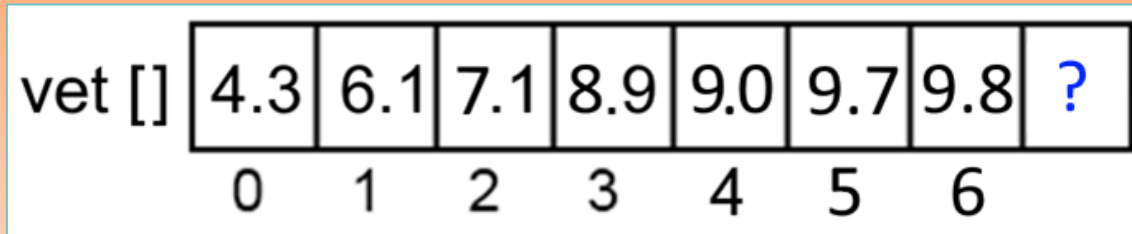
Deslocamentos em vetores

Obs: Em algumas linguagens de programação poderiam existir classes prontas que permitam inserir e eliminar itens em qualquer posição de um vetor, efetuando o redimensionamento do vetor e os deslocamentos necessários em forma automática.

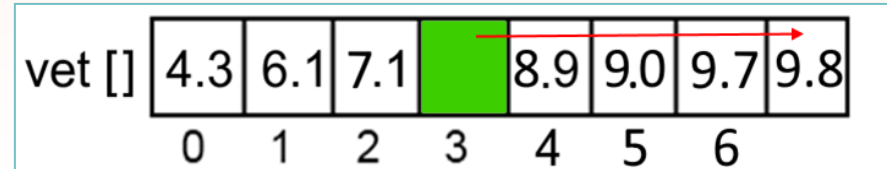
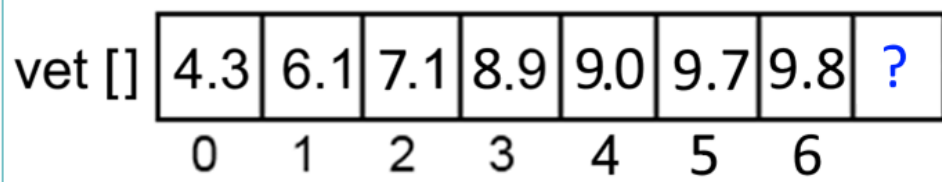
Em Java temos as classes `ArrayList` (não sincronizada) e `Vetor` (sincronizada, thread-safe) que atendem os requisitos anteriores.

Deslocamento de vetores

- São situações onde um trecho do vetor será deslocado para direita ou para esquerda. Por exemplo, ao **eliminar** um item (o item 7.1 na figura), deslocaremos os restantes **para a esquerda**:

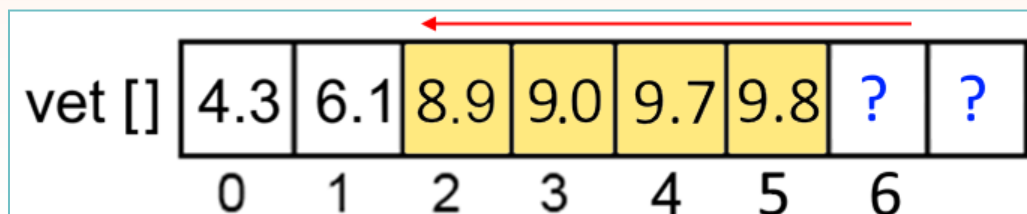
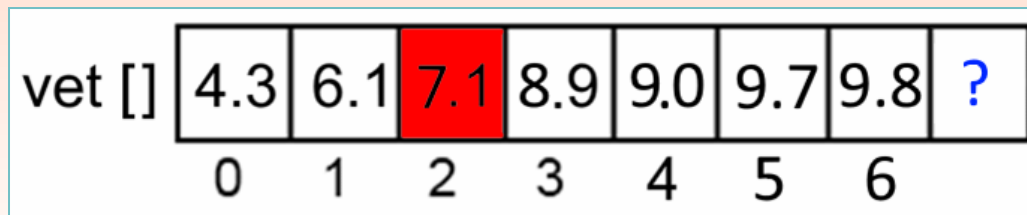


- Por exemplo, para **inserir** um novo item em uma posição, deslocaremos os itens restantes **para a direita** (usando a memória livre do vetor):



Deslocamento de vetores (uma versão simplificada)

```
void deslocaParaEsquerda (double vet[], int de, int ate)
{
    // eliminamos o item na posição (de-1) e deslocamos
    // os restantes para a esquerda
    for (int i = (de - 1); i < ate; i++) {
        vet[i] = vet[i+1];
    }
    vet[ate] = 0; // só para marcar o item final
}
```



Exemplo: deslocar para esquerda os itens entre as posições 3 e 6, então:

de=3, ate=6

Deslocamento de vetores (uma versão simplificada)

// Obs: na chamada destes algoritmos talvez melhor não utilizar *vet.length*.
// Utilize uma variável com a quantidade verdadeira de itens
// (para vetores sem ocupação total).

```
void deslocaParaDireita (double vet[], int de, int ate)  
{  
    // Abrimos um espaço e ocupamos uma posição no final  
    // i irá diminuindo, observe o i--  
    for (int i = ate; i >= de; i--) {  
        vet[i+1] = vet[i];  
    }  
    vet[de] = 0; // só para marcar o item "vazio"  
}
```

| | | | | | | | | |
|--------|-----|-----|-----|-----|-----|-----|-----|---|
| vet [] | 4.3 | 6.1 | 7.1 | 8.9 | 9.0 | 9.7 | 9.8 | ? |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| | | | | | | | | |
|--------|-----|-----|-----|---|-----|-----|-----|-----|
| vet [] | 4.3 | 6.1 | 7.1 | | 8.9 | 9.0 | 9.7 | 9.8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Exemplo: deslocar para direita os itens entre as posições 3 e 6, então:

de=3, **ate**=6

Deslocamento de vetores (versões mais completas)

```
void deslocaDireita (double vet[], int de, int ate) {  
    //alguns testes de pré-requisitos permitem evitar erros:  
    if(de > ate)return;  
    if(de < 0)de=0;  
    if(ate > vet.length-2)ate=vet.length-2;  
    // abre um espaço na posição 'de' e ocupamos mais uma posição no final  
    for (int i = ate; i >= de; i--) vet[i+1] = vet[i]; //obs: ciclo começando no final  
    vet[de] = 0; // só para marcar o item que ficou "vazio"  
}
```

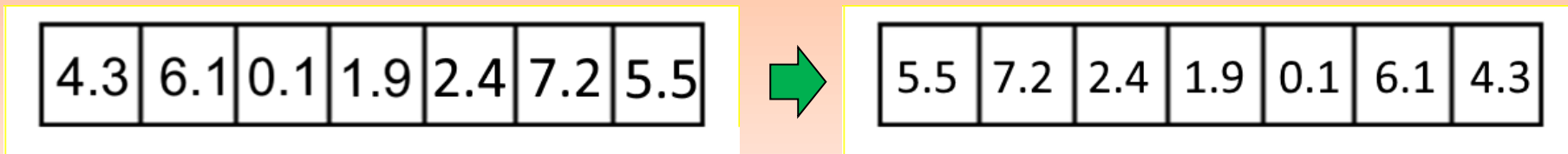
```
void deslocaEsquerda (double vet[], int de, int ate) {  
    //alguns testes de pré-requisitos para evitar erros:  
    if(de > ate)return;  
    if(de <= 0)de=1; //corrigimos a posição, mas poderíamos abortar  
    if(ate > vet.length-1)ate=vet.length-1; //corrigimos a posição, mas...  
    // elimina o item na posição (de-1) e desloca os restantes para a esquerda  
    for (int i = (de - 1); i < ate; i++) vet[i] = vet[i+1];  
    vet[ate] = 0; // só para marcar o item final  
}
```

Exemplo no projeto **Buscas**

Exercício 1 (ver figura no próximo slide)

Elabore um método (e os testes necessários) para inverter completamente os elementos guardados em um vetor. Considere uma variável N que armazena a quantidade de elementos reais do vetor e não seu tamanho alocado. Visualize o vetor depois de invertido.

```
public void inverter(double vet[], int N)
```



Exercício 2

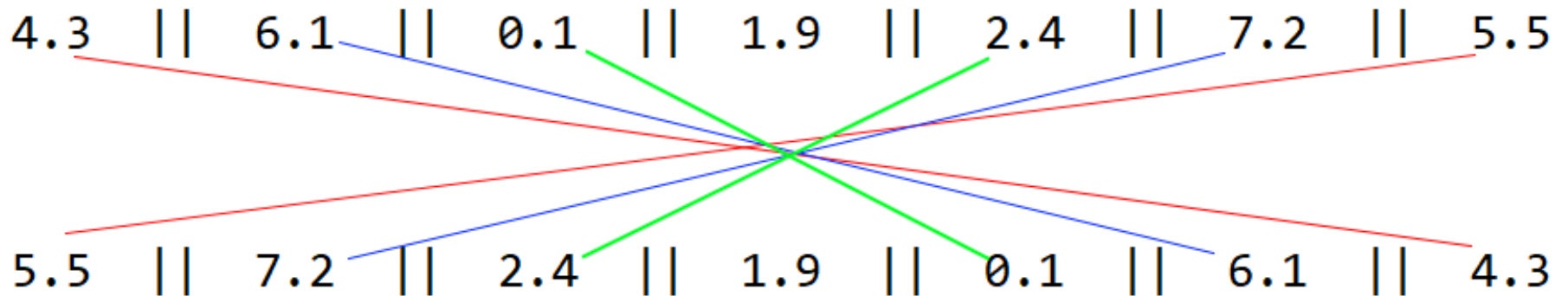
Elabore um método (e os testes necessários) para inverter completamente os elementos guardados em um ArrayList. Visualize o ArrayList depois de invertido.

```
public void inverter(ArrayList arr)
```

Exercícios

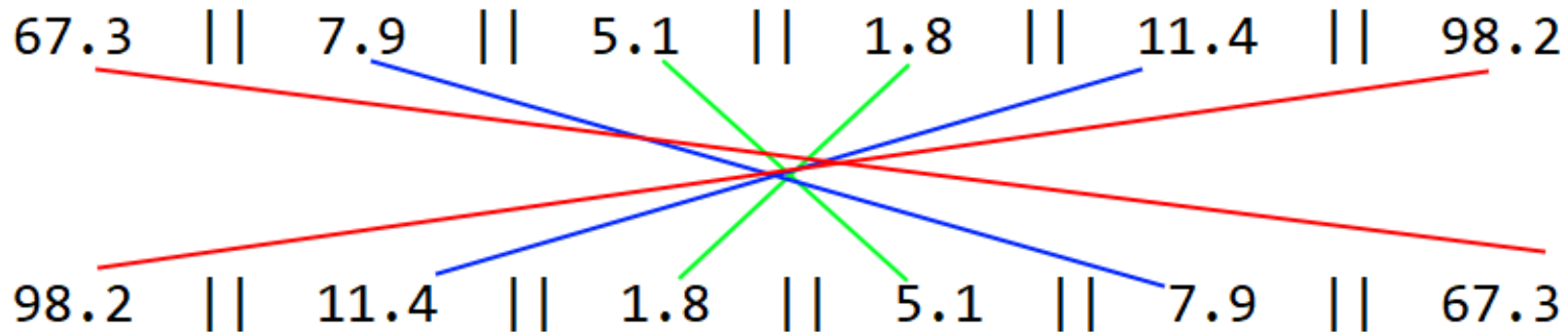
----- Vetor com qtde. elementos ímpar -----

4.3 || 6.1 || 0.1 || 1.9 || 2.4 || 7.2 || 5.5
5.5 || 7.2 || 2.4 || 1.9 || 0.1 || 6.1 || 4.3



----- Vetor com qtde. elementos par -----

67.3 || 7.9 || 5.1 || 1.8 || 11.4 || 98.2
98.2 || 11.4 || 1.8 || 5.1 || 7.9 || 67.3



Bibliografia (oficial) para a disciplina

| BIBLIOGRAFIA BÁSICA | BIBLIOGRAFIA COMPLEMENTAR |
|--|--|
| CORMEN, T. H.; et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012. | ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010. (eBook) |
| GOODRICH, M. T.; TAMASSIA, R. Estrutura de dados e algoritmos em java. 5. ed. Porto Alegre: Bookman, 2013. (livro físico e e-book) | PUGA, S.; RISSETTI, G. Lógica de programação e estruturas de dados, com aplicações em Java. 3. ed. São Paulo: Pearson Education do Brasil, 2016. (eBook) |
| CURY, T. E., BARRETO, J. S., SARAIVA, M. O., et al. Estrutura de Dados (1. ed.) ISBN 9788595024328, Porto Alegre: SAGAH, 2018 (e-book) | DEITEL, P.; DEITEL, H. Java como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017. (eBook) |
| | BARNES, D. J.; KOLLING, M. Programação Orientada a Objetos com Java: uma introdução prática usando o Blue J. São Paulo: Pearson Prentice Hall, 2004. (eBook) |
| | BORIN, V. POZZOBON. Estrutura de Dados. ISBN: 9786557451595, Edição: 1ª . Curitiba: Contentus, 2020 (e-book) |