02

Implementando o intercala em um único array

Transcrição

Nós vimos que, na prática, teremos que suportar uma maneira de intercalar os elementos mesmo que eles não venham em dois *array*s distintos. É possível que eles venham em um único *array*, em que a primeira e a segunda partes foram ordenadas do menor para o maior. Nós iremos intercalar estas partes. Vamos implementar isto?

Iremos criar uma nova classe, chamada de `TestaIntercalaEmUmArray`, que colocará o método `main` para criar todas as notas que havíamos feito anteriormente.

```
public static void main(String[] args) {
    Nota[] notasDoMauricio = {
        new Nota("andre", 4)
        new Nota("mariana", 5),
        new Nota("carlos", 8.5),
        new Nota("paulo", 9)
};

    Nota[] notasDoAlberto = {
        new Nota("jonas", 3),
        new Nota("juliana", 6.7),
        new Nota("guilherme", 7),
        new Nota("lucia", 9.3),
        new Nota("ana", 10)

        };
}
```

Porém, em vez de trabalharmos com duas variáveis, teremos apenas uma. Primeiro a do Maurício, depois a do Alberto. Vamos substituir `notasDoMauricio` por `notas`.

```
public static void main(String[] args) {
    Nota[] notas = {
        new Nota("andre", 4)
        new Nota("mariana", 5),
        new Nota("carlos", 8.5),
        new Nota("paulo", 9)
        new Nota("jonas", 3),
        new Nota("juliana", 6.7),
        new Nota("guilherme", 7),
        new Nota("lucia", 9.3),
        new Nota("ana", 10)

        };
}
```

Temos as notas, agora nosso objetivo é intercalar todas elas. O código que iremos escrever, ficará muito parecido com o `rank` da classe `TestaMerge`. Nós iremos dizer intercala as notas e depois as imprima.

```
Nota[] rank = intercala(notas);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
}
```

Falta ainda implementar a função `intercala()` que recebe o *array* único, em contraste a que recebia dois *array*s. Vou criar o método `intercala`, que receberá as notas. Qual será o tamanho do *array* de resultado? Ele será do mesmo tamanho do `notas.length`. No entanto, nós não precisaremos ficar somando os dois, porque já estará tudo em um *array*. E o programa irá retornar o próprio `resultado`.

```
private static Nota[] intercala(Nota[] notas) {
    Nota[] resultado = new Nota[notas.length];
    return resultado;
}
```

O que queremos fazer em seguida é a intercalação. Antes, nós tínhamos três variáveis para acompanhar os nosso passos. Uma era `atual`, aonde colocaríamos o próximo elemento. O `atual1`, que era o acompanhamento do *array* da esquerda. E o `atual2`, que era o acompanhamento do *array* da direita. As duas variáveis começavam com 0.

```
private static Nota[] intercala(Nota[] notas) {
    Nota[] resultado = new Nota[notas.length];
    int atual = 0;
    int atual1 = 0;
    int atual2 = 0;
    return resultado;
}
```

As duas começavam por esta posição, porque estavam separados. No entanto, unimos todos os elementos em um único *array* e o `atual2` irá começar a partir da posição 4 (o Jonas) .

```
Nota[] notas = {
        new Nota("andre", 4)
        new Nota("mariana", 5),
        new Nota("carlos", 8.5),
        new Nota("paulo", 9)
        new Nota("jonas", 3),
        new Nota("juliana", 6.7),
        new Nota("guilherme", 7),
        new Nota("lucia", 9.3),
        new Nota("ana", 10)

        };
```

Como conseguiremos que o `atual2` saiba que ela precisa começar pela posição 4? Quando criarmos o método intercala, precisamos determinar que ele intercale a partir da posição 4.

```
Nota[] rank = intercala(notas, 4);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
}
```

A primeira parte, que segue do 0 até o 3, é referente ao lado da esquerda. Do 4 até o fim, será a parte da direita. Podemos deixar isto explicito: do 0 até o 4 exclusive, será a parte da esquerda.

```
Nota[] rank = intercala(notas, 0, 4);
```

Do 4 em diante, o que significa até o `notas.length` será a parte da direita.

```
Nota[] rank = intercala(notas, 0, 4, notas.length);
```

Assim deixamos bem explícito qual é a parte referente ao lado esquerdo e ao lado direito. Indicamos o `int inicial` na esquerda, o `int miolo` , que é o ponto que separa as partes, e o `int termino` na direita.

```
private static Nota[] intercala(Nota[] notas, int inicial, int miolo, int termino) {
    Nota[] resultado = new Nota[notas.length];
    int atual = 0;
    int atual1 = 0;
    int atual2 = 0;
    return resultado;
}
```

Os valores da variáveis serão: `atual` é 0, `atual1` é o `inicial` e o `atual2 = miolo` .

```
private static Nota[] intercala(Nota[] notas, int inicial, int miolo, int termino) {
    Nota[] resultado = new Nota[notas.length];
    int atual = 0;
    int atual1 = inicial;
    int atual2 = miolo;
    return resultado;
}
```

Agora podemos trabalhar tanto com a parte da esquerda como a da direita, com o `atual1` e o `atual2` .

Em seguida, iremos incluir o laço. Enquanto (`while`) o `atual1` for menor do que o tamanho do *array* da esquerda (o `miolo`) e o `atual2` for menor que o `termino` , nos seguiremos andando com as variáveis.

```
private static Nota[] intercala(Nota[] notas, int inicial, int miolo, int termino) {
    Nota[] resultado = new Nota[notas.length];
    int atual = 0;
    int atual1 = inicial;
    int atual2 = miolo;
    while(atual1 < miolo &&
          atual2 < termino) {
```

```
    }
      return resultado;
  }
```

Iremos verificar alguns dados. `miolo` é igual a 4, então o indicador da direita já deve começar nesta posição. Ele inicia com o Jonas. O indicador da esquerda começa pelo 0, que é o André. Observe que quando `atual1` for igual 4, e for referenciar o Jonas, ele não pode continuar. A condiçao é que o `atual1` seja menor que o `miolo` e não menor ou igual(`<=`). Nós já haviamos feito o mesmo antes, no `TestaMerge` . O `atual1` era menor do que o lado esquerdo (`notas1.length`).

```
  while(atual1 < notas1.length &&
        atual2 < notas2.length){
    }
```

O mesmo foi feito para o `atual2` . Então, iremos determinar isto no `while` do `TestaIntercalaEmUmArray` . `atual2` será menor do que do que o fim do lado direito (`termino`).

```
  while(atual1 < miolo &&
            atual2 < termino) {

    }
```

O laço irá passar por todos os elementos. O que mais precisamos fazer? Comparar as duas notas, a `notas1` , que é igual a `notas[atual1]` , e `notas2` , que é igual a `notas[atual2]` . Vamos escrever o `if` : se a `nota1.getValor` for igual ao `notas2.getValor` , o `resultado[atual]` será igual a `nota1` . Senão (`else`), o `resultado[atual]` será igual a `nota2` . Incluímos as duas notas.

```
  if(nota1.getValor() < valor2.getValor()) {
      resultado[atual] = nota1;
  } else {
      resultado[atual] = nota2;
  }
```

Independente do caso iremos somar +1 ao `atual` . Porém, na esquerda iremos somar +1 no `atual1` e na direita somaremos +1 no `atual2` .

```
  if(nota1.getValor() < valor2.getValor()) {
      resultado[atual] = nota1;
      atual1++;
  } else {
      resultado[atual] = nota2;
      atual2++;
  }
  atual++;
```

Sabemos que não basta comparar, enquanto tivermos dois elementos, porque quando termina um dos *array*s irá sobrar itens no outro. Precisaremos também dos dois `while` s no fim. Enquanto (`while`) o `atual1` for menor do que o `miolo` , precisamos que todos os elementos sejam copiados. Por isso, `resultado[atual]` será igual a `notas[atual1]` . Depois somaremos +1 no `atual1` e no `atual` .

```
while(atual1< miolo) {
    resultado[atual] = notas[atual1];
    atual1++;
    atual++;
}
```

Faremos o mesmo para o `atual2`. O `while` irá indicar que enquanto ele for menor do que o `termino`, o `resultado[atual]` será igual a `notas[atual2]`. E somaremos +1 no `atual2` e no `atual`.

```
while(atual2< termino) {
    resultado[atual] = notas[atual2];
    atual2++;
    atual++;
}
```

Nós traduzimos o método que recebia dois *array*s, que estavam totalmente separados. Porém, sabemos que na prática, os elementos virão dispostos em um *array* único. Ao observá-lo, reconhecemos onde existe uma quebra entre as parte. E precisamos ordenar de uma parte até a outra, intercalando os elementos.

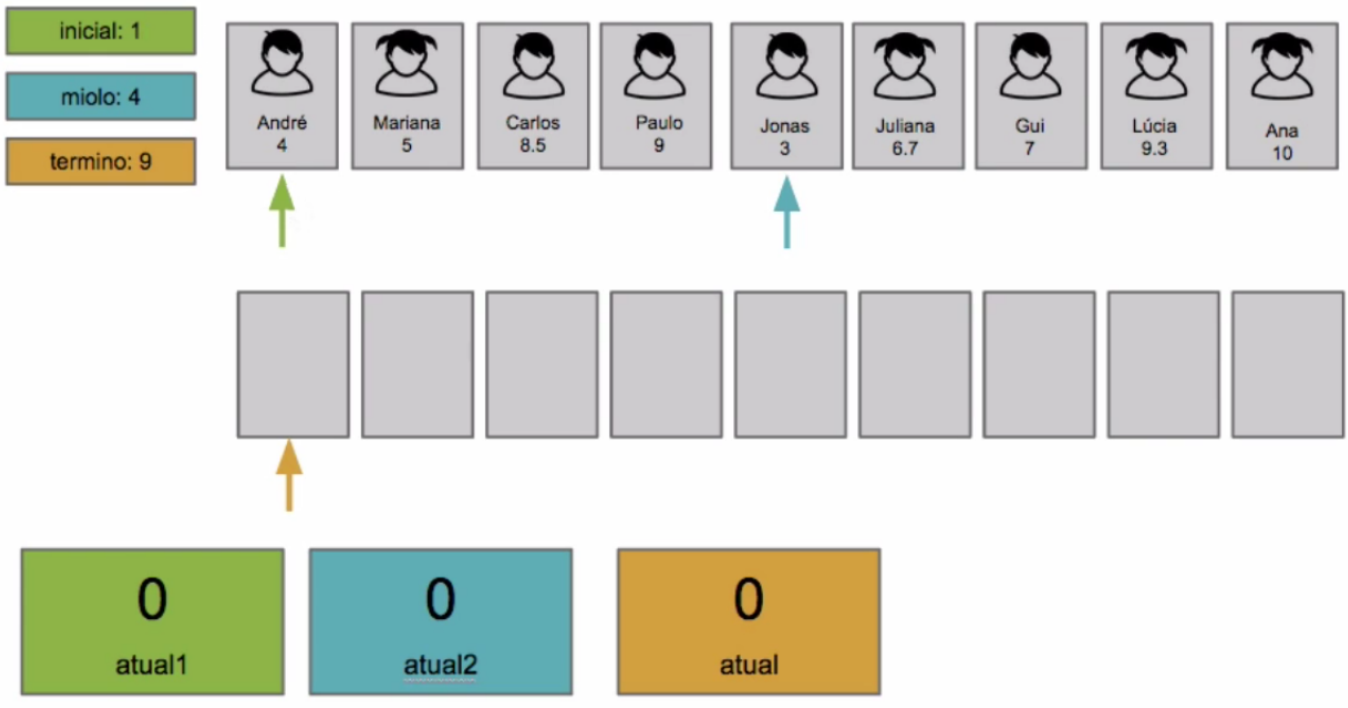
Ao testarmos o código, o resultado será:

```
- jonas 3.0
- andre 4.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
- carlos 8.5
- paulo 9.0
- lucia 9.3
- ana 10.0
```

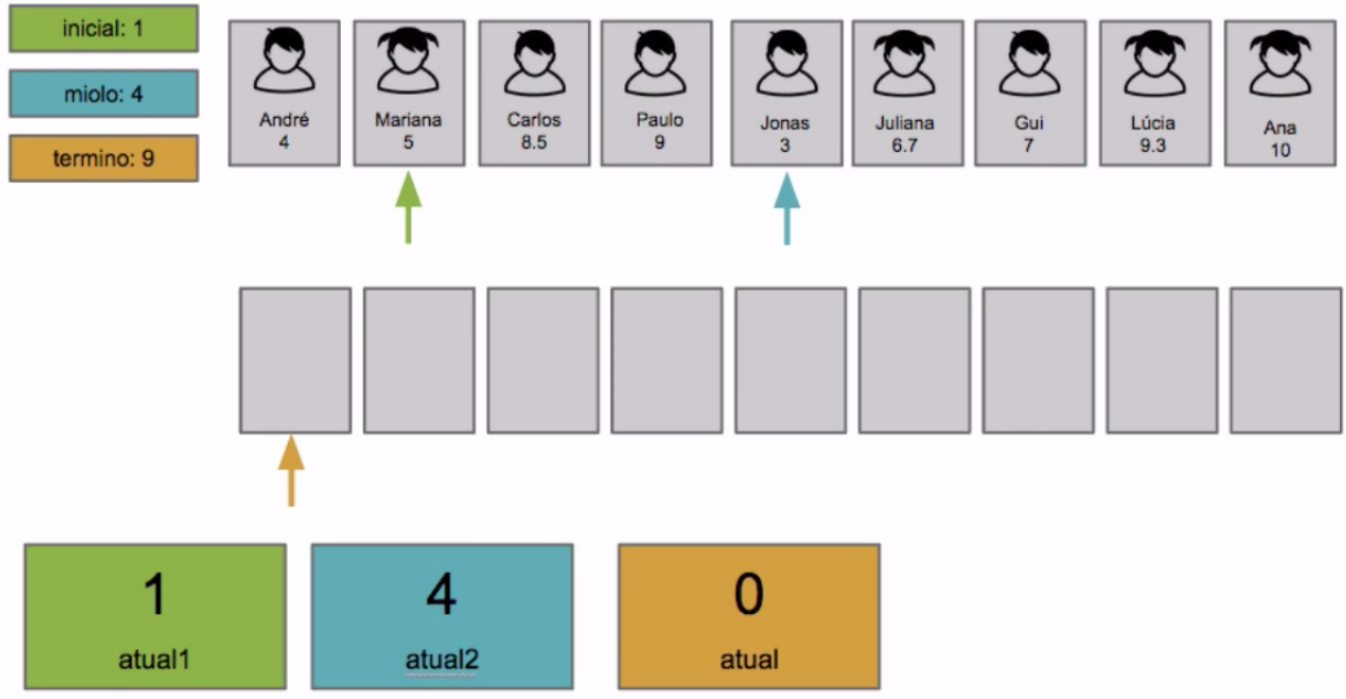
Os elementos do nosso *array* foram intercalados corretamente.

Simulando o intercala em um único array

Agora temos uma função que intercala. Ela parece funcionar corretamente, mas vamos testá-la. Nós já a testamos com as variáveis `inicial` igual 0, `miolo` igual 4 e `termino` igual 9. Porém, o que acontecerá se quisermos manter o primeiro elemento, o André, no inicio da lista. Imagine os seguintes exemplos: deixaremos a prova de um aluno no início da pilha, porque queremos conversar com ele... Ou queremos deixar a carta do *joker* (o coringa) no alto do monte de baralho. Então, não iremos começar a intercalação a partir da posição 0, mas da 1. Enquanto, `inicial` será igual a 1.

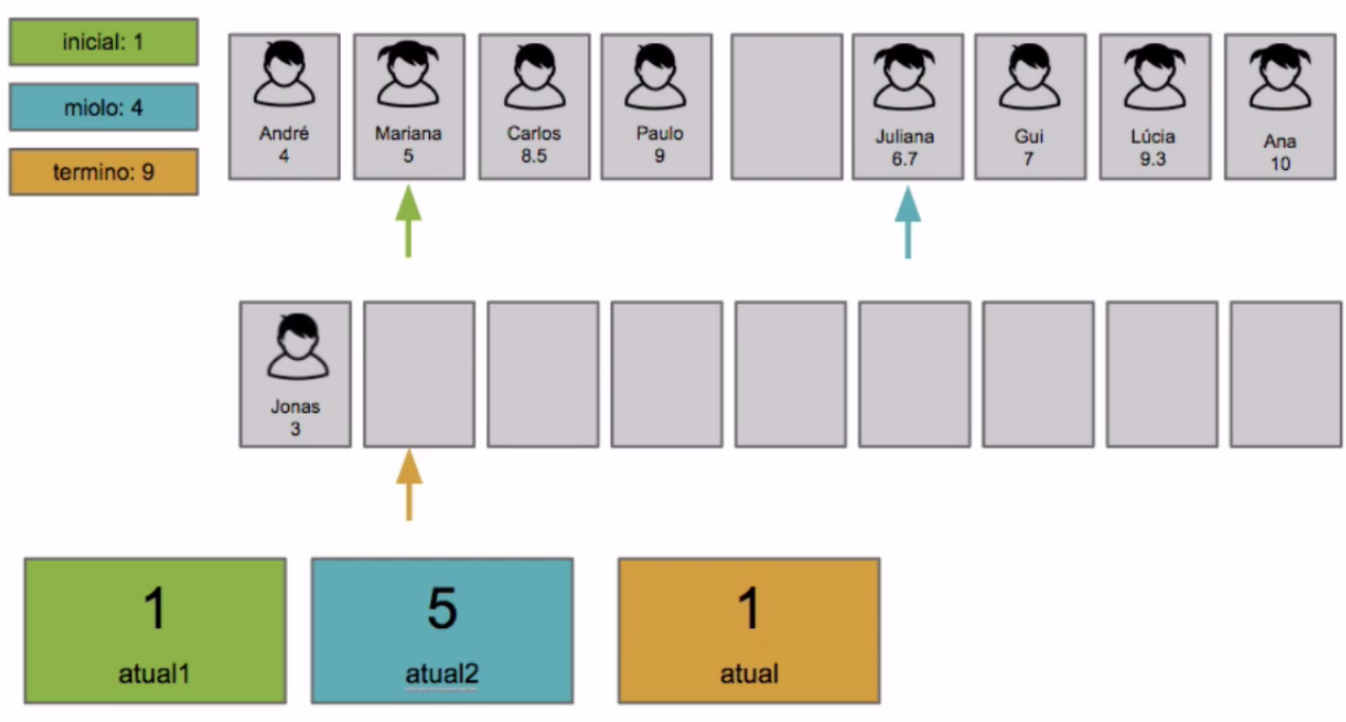


Iremos iniciar com a Mariana. As variáveis terão os seguintes valores: `atual1` será igual 1, `atual2` igual a 4 e `atual` igual a 0.

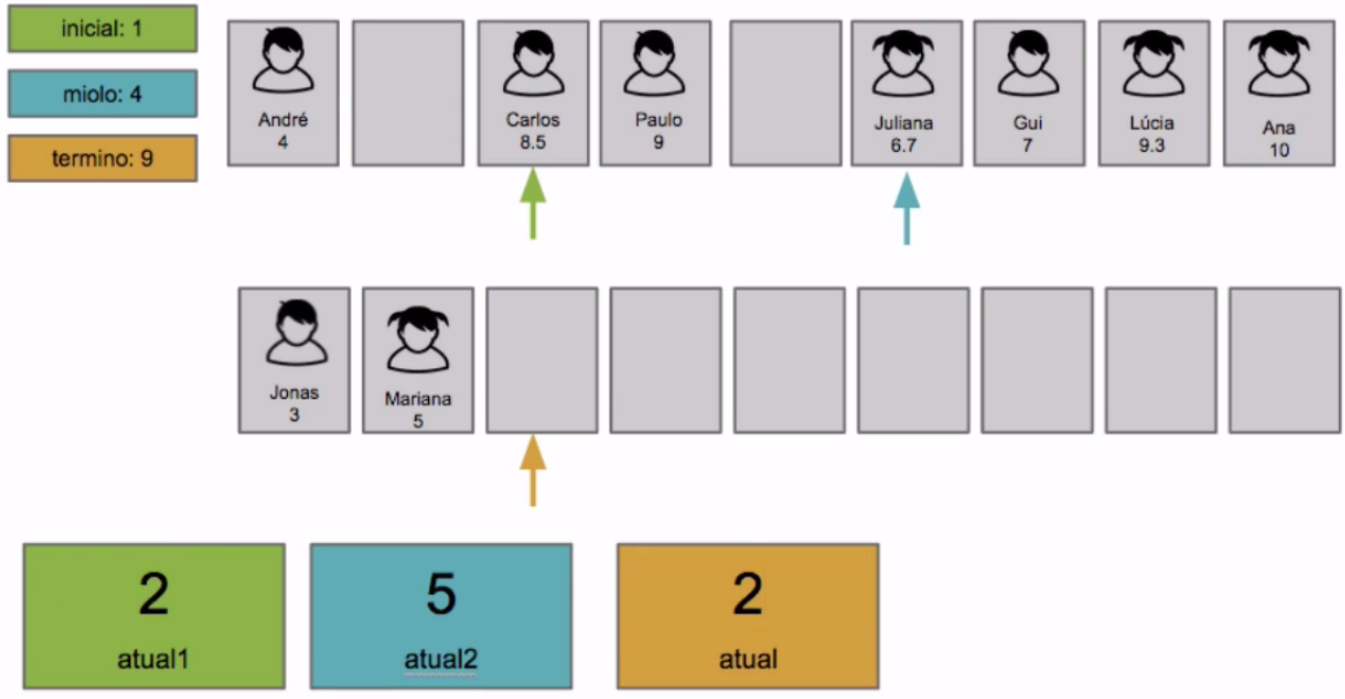


Vamos rodar o algoritmo e ver o que acontece.

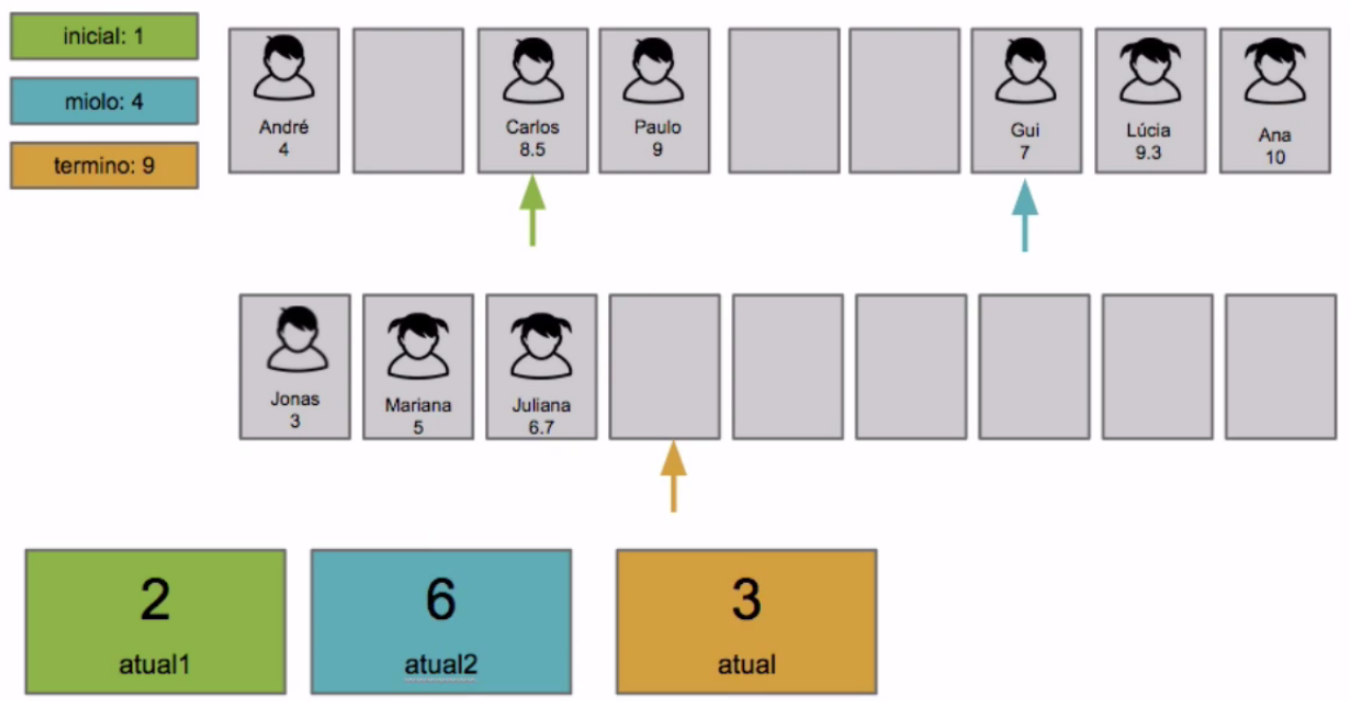
Ao compararmos os primeiros elementos, identificamos que Jonas é o menor. Iremos movê-lo para o novo *array*. Também modificaremos as posições dos indicadores e os valores das variáveis. `atual2` será igual a 5 e o `atual` igual a 1.



Entre Mariana e Juliana, qual é a menor? A Mariana. Iremos colocá-la no *array*. Vamos alterar os valores da variáveis: `atual1` será igual a 2 e `atual` será 2.

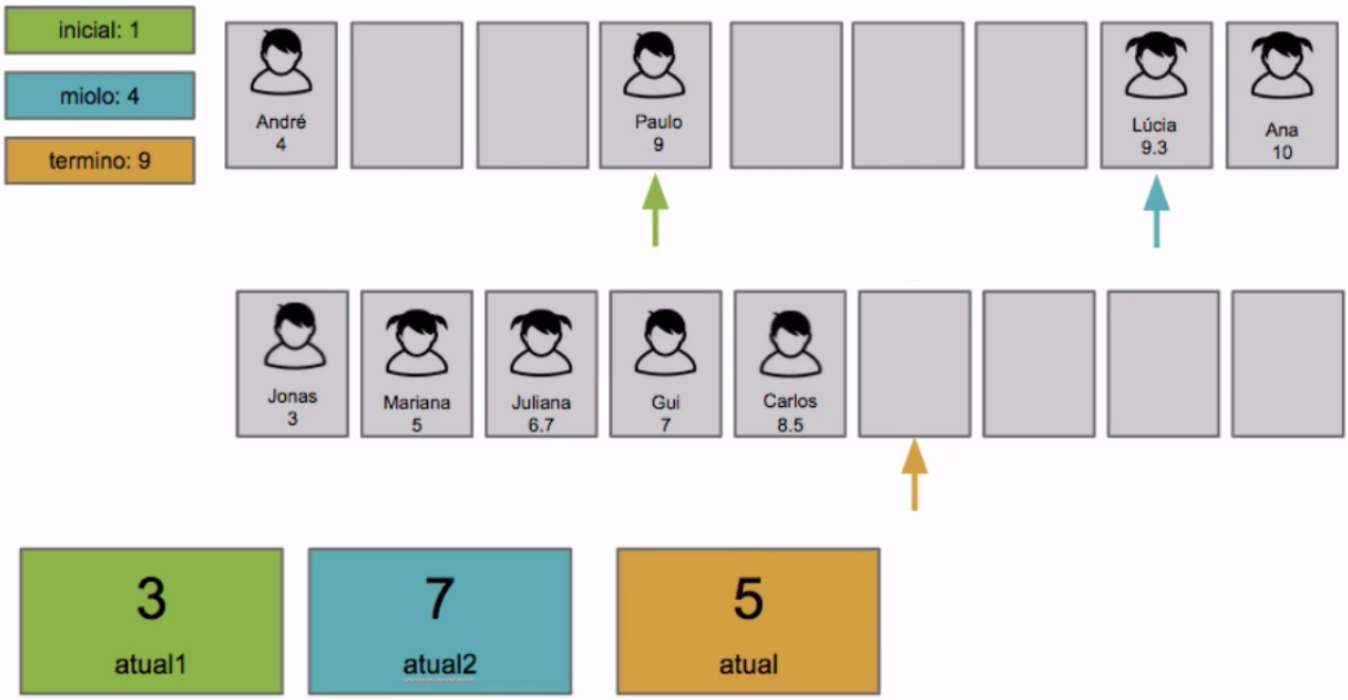


Comparando Carlos e Juliana, qual é o menor? A Juliana. É a vez dela de ser movida. Alteramos os valores das variáveis e `atual2` será igual a 6, enquanto `atual` será igual a 3.

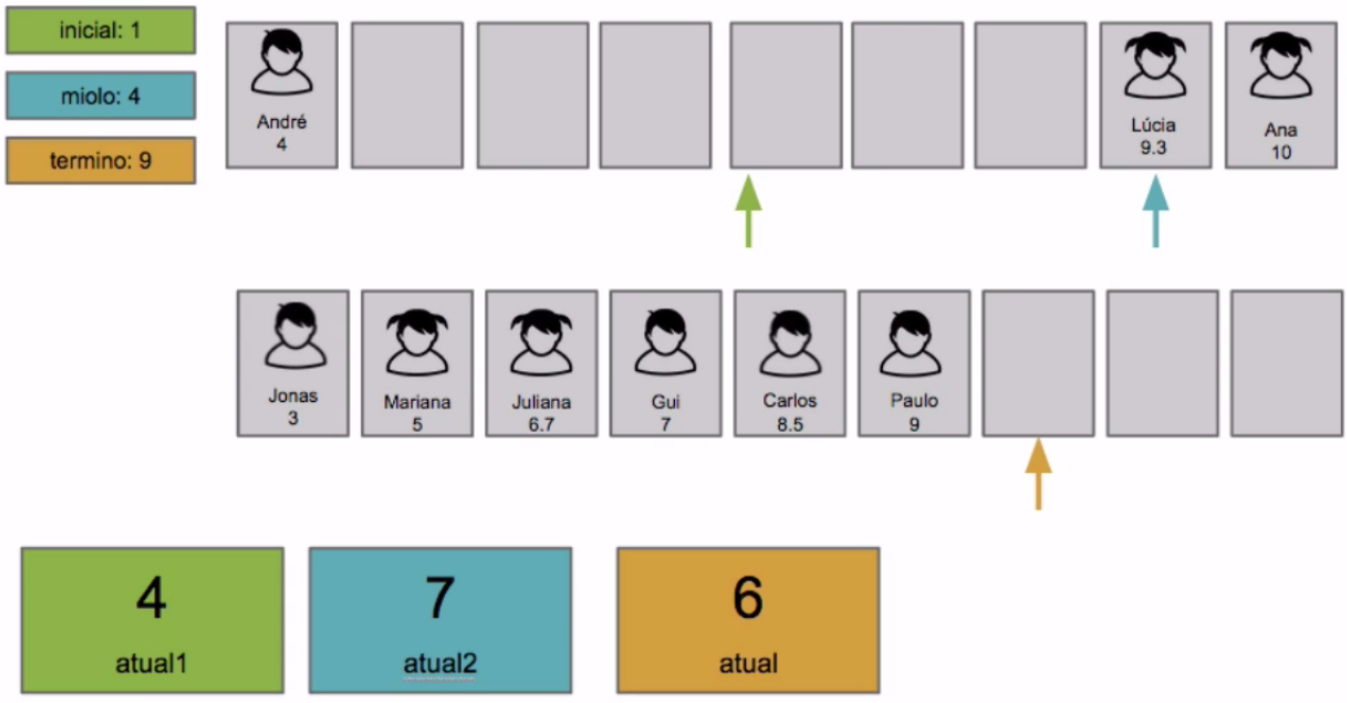


Compararemos o Carlos e o Gui. Qual deles é o menor? É o Gui. Iremos movê-lo para o novo *array*. Andamos com os indicadores e a variável `atual2` será igual a 7, enquanto `atual` será igual a 4.

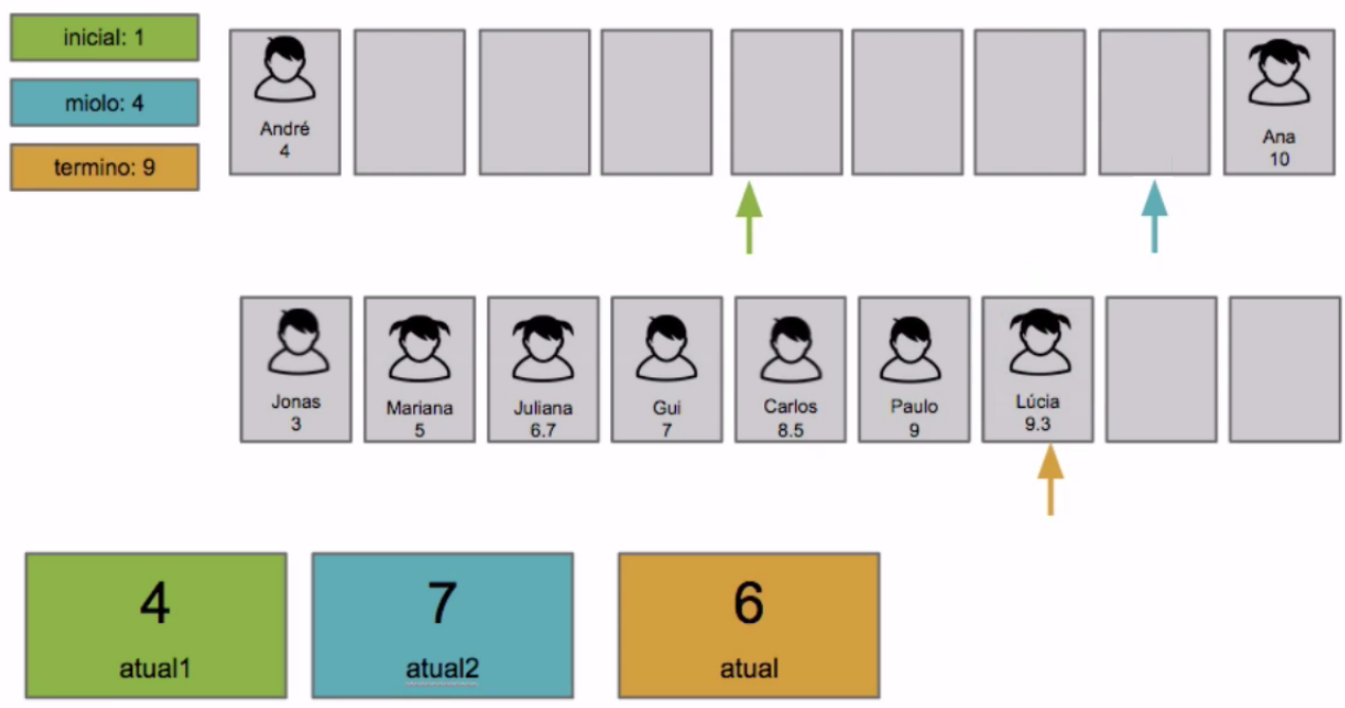
Entre Carlos e Lúcia, qual é o menor? O Carlos. Vamos movê-lo para o *array*. Seguimos alterando os valores das variáveis: `atua1` será igual a 3 e `atual` igual a 5.



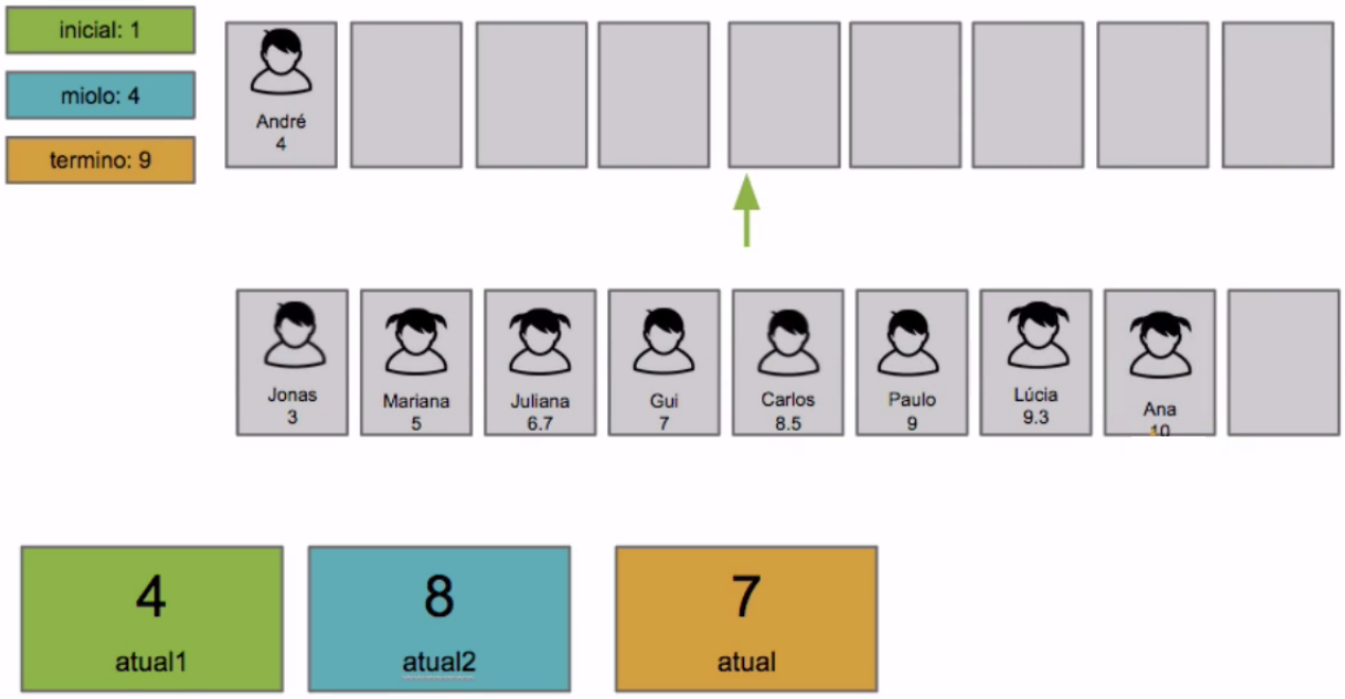
Entre Paulo e Lúcia, qual é o menor elemento? É o Paulo. Ele é o próximo a ser inserido no *array*. Iremos alterar a posição dos indicadores e os valores das variáveis: `atual3` será igual 4 e `atual` igual a 6.



Nosso *array* da esquerda acabou, afinal `atual1` é igual ao valor do `miolo`. O que devemos fazer? Apenas copiar os elementos que sobraram no outro *array*. Primeiro moveremos a Lúcia, assim como as posições dos indicadores também serão modificadas.



Depois, mudamos os valores das variáveis e `atual2` será igual a 8 e `atual` será igual a 7. Ainda temos algum elemento para ser copiado? Sim, a Ana. Iremos movê-la para o novo *array*.



Vamos alterar o `atual` para 8 e o `atual2` será igual a 9. A casa que sobra no novo *array* ficará vazia, porque um dos elementos do *array* da esquerda permaneceu lá. Terminamos a ordenação.