03

Reduzindo o código

Transcrição

Teremos que diminuir bastante o código para ser aprovado no Spoj. Não havíamos notado que ao final do enunciado temos um desafio (*chalenge*): deixar o código pequeno. O que faremos agora é diminuir nosso código em bytes. E isso não tem nada a ver com qualidade, funcionalidade, otimização do código, programar menos ou facilitar a manutenção; é apenas um desafio. Inclusive, todos esse itens costumam piorar em muitas situações. Mas vamos analisar parte por parte. Dê mais uma olhada na nossa função `ehPalindromo` .

```
private static boolean ehPalindromo(String palavra) {
  //System.out.println(palavra + " " + palavra.lenght());

  boolean ehPalindromo = true;
  for(int i = 0; i < palavra.length(); i++){
    char esquerda = palavra.charAt(i);
    char direita = palavra.charAt(palavra.lenght() - 1 -i);

    //System.out.println(esquerda + " " + direita);
    if(esquerda == direita){
      //feliz
    } else {
        ehPalindromo = false;
    }

  }
  return ehPalindromo;
```

Vamos pensar na definição de palíndromo: algo que se lê igual da direita para a esquerda e da esquerda para a direita. A leitura que fizemos comprova isso.

```
Alura-Azul:bin alura$ java Main < entrada1.txt
1 HELloollEH 10
"YES"
2 ILOVEyou 8
"NO"
Alura-Azul:bin alura$ java Main < entrada1.txt
1 HELloollEH 10
H H
E E
L L
l l
o o
o o
l l
L L
E E
H H
"YES"
2 ILOVEyou 8
I u
```

```
L o
O y
V E
E V
y O
o L
u i
"NO"
```

Quando o primeiro caractere já não é igual ao último, a palavra não será um palíndromo. Ou seja, poderíamos parar de ler na primeira letra que divergisse. Assim, no `else` já podemos colocar `return false`.

```
private static boolean ehPalindromo(String palavra) {
  //System.out.println(palavra + " " + palavra.lenght());

  boolean ehPalindromo = true;
  for(int i = 0; i < palavra.length(); i++){
    char esquerda = palavra.charAt(i);
    char direita = palavra.charAt(palavra.lenght() - 1 -i);

    //System.out.println(esquerda + " " + direita);
    if(esquerda == direita){
      //feliz
    } else {
        return false;
    }

  }
  return ehPalindromo;
```

E se já sabemos que não é palíndromo antecipadamente, a leitura só chagar ao final. Ou seja, o último `return` deve ser `true`, e não precisaremos do `boolean` do início.

```
private static boolean ehPalindromo(String palavra) {
  //System.out.println(palavra + " " + palavra.lenght());

  for(int i = 0; i < palavra.length(); i++){
    char esquerda = palavra.charAt(i);
    char direita = palavra.charAt(palavra.lenght() - 1 -i);

    //System.out.println(esquerda + " " + direita);
    if(esquerda == direita){
      //feliz
    } else {
        return false;
    }

  }
  return true;
```

Com essa mudança, perceba que o `else` já não precisa existir, se mudarmos o `if()` para `if(esquerda =! direita)`.

```
  private static boolean ehPalindromo(String palavra) {
    //System.out.println(palavra + " " + palavra.lenght());
    for(int i = 0; i < palavra.length(); i++){
      char esquerda = palavra.charAt(i);
      char direita = palavra.charAt(palavra.lenght() - 1 -i);
      //System.out.println(esquerda + " " + direita);
      if(esquerda =! direita){
          return false;
      }

    }
    return true;
```

Vamos testar no terminal:

```
Alura-Azul:bin alura$ java Main < entrada1.txt1
1 "YES"
2 "NO"
```

Continua funcionando. Vamos imprimir a leitura letra a letra para ver como ela está acontecendo?

```
  private static boolean ehPalindromo(String palavra) {
    System.out.println(palavra + " " + palavra.lenght());
    for(int i = 0; i < palavra.length(); i++){
      char esquerda = palavra.charAt(i);
      char direita = palavra.charAt(palavra.lenght() - 1 -i);
      //System.out.println(esquerda + " " + direita);
      if(esquerda =! direita){
          return false;
      }

    }
    return true;
```

No terminal:

```
Alura-Azul:bin alura$ java Main < entrada1.txt1
1 "YES"
2 "NO
Alura-Azul:bin alura$ java Main < entrada1.txt1
1 H H
E E
L L
l l
o o
o o
l l
L L
E E
H H
"YES"
```

```
2 I u
"NO"
```

Vemos que em `ILOVEyou`, a leitura foi interrompida na primeira letra e o `"NO"` já aparece. Olhemos novamente para o `HELloolLEH`. Se lermos apenas até a metade, inclusive a letra do meio, já teremos coberto a palavra toda. Podemos eliminar o entrecruzamento das leituras dos dois sentidos. Em muitos algoritmos não precisamos ir até o final e não faz diferença pararmos na metade. Mas há casos em que faz diferença dividir o tempo por dois. Talvez não faça diferença nesse caso, sequer é a questão que precisamos resolver agora. Precisamos diminuir o código, mas podemos acrescentar que veremos só até o `palavra.lenght() / 2`.

```
    private static boolean ehPalindromo(String palavra) {
      System.out.println(palavra + " " + palavra.lenght());
      for(int i = 0; i < palavra.length() / 2; i++){
        char esquerda = palavra.charAt(i);
        char direita = palavra.charAt(palavra.lenght() - 1 -i);
        //System.out.println(esquerda + " " + direita);
        if(esquerda =! direita){
            return false;
        }

      }
      return true;
```

Isso é: se o comprimento era `10`, podemos ler até o quinto algarismo, ou a posição `i = 4`. Para o `HELloolLEH` deve dar certo. Vamos conferir no terminal:

```
Alura-Azul:bin alura$ java Main < entrada1.txt
1 "YES"
2 "NO
Alura-Azul:bin alura$ java Main < entrada1.txt
1 H H
E E
L L
l l
o o
"YES"
2 I u
"NO"
```

Agora vamos testar em um caso que o enunciado não cita; uma palavra com número ímpar de letras. Criaremos a `entrada3.txt`, que será:

```
3
HELloolLEH
HELlolLEH
ILOVEyou
```

A segunda entrada é quase igual à primeira, mas repetimos o uma versão da primeira palavra com um `o`. Ele também é palíndromo, afinal, o `o` é igual a si mesmo. Vamos testar a leitura dessa entrada?

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 H H
E E
L L
l l
o o
"YES"
2 H H
E E
L L
l l
"YES"
3 I u
"NO"
```

Na versão com letras ímpares, o programa sequer leu a letra do meio. Assim, no nosso algoritmo, não há problema em manter a linha `for(int i = 0; i < palavra.length() / 2; i++){` com o símbolo de `<`. Em outros casos, poderíamos precisar de um `<=`. É preciso que sempre pensemos nas bordas. Vamos colocar o `sysout` que mostra letra a letra como comentário novamente.

```
    private static boolean ehPalindromo(String palavra) {
        //System.out.println(palavra + " " + palavra.lenght());
        for(int i = 0; i < palavra.length() / 2; i++){
          char esquerda = palavra.charAt(i);
          char direita = palavra.charAt(palavra.lenght() - 1 -i);
          //System.out.println(esquerda + " " + direita);
          if(esquerda =! direita){
              return false;
          }

        }
        return true;
```

E testaremos novamente.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
```

Podemos verificar quantos caracteres temos aqui com um programinha do Linux chamado `wc`. Como estamos no diretorio `bin`, precisamos dar um `pwd` para entrar na pasta certa.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
Alura-Azul:bin alura$ pwd
/Users/alura/guilherme/Documents/workspace/palincod/bin
Alura-Azul:bin alura$ wc ../src/Main.java
      44      108      859 ../src/Main.java
Alura-Azul:bin alura$
```

Ele nos mostra que o nosso código tem `859` caracteres, e precisamos deixá-lo com `250`. Vai ser difícil cumprir esse desafio em Java. Vamos começar a diminuir o código, vendo o que é possível cortar.

Até então estávamos melhorando o algoritmo. Lembrando a definição de palíndromo: ler da esquerda para a direita é igual a ler da direita para a esquerda. Mas estamos verificando isso caractere a caractere. Isto é: ler a palavra inteira na memória e comparar as letras uma a uma, e ao chegar na metade, ele para.

```
private static boolean ehPalindromo(String palavra) {
    //System.out.println(palavra + " " + palavra.lenght());
    for(int i = 0; i < palavra.length() / 2; i++){
      char esquerda = palavra.charAt(i);
      char direita = palavra.charAt(palavra.lenght() - 1 -i);
      //System.out.println(esquerda + " " + direita);
      if(esquerda =! direita){
          return false;
      }

    }
    return true;
```

Será que existe algum método de string que inverte a palavra? Algo como `reverse` ou `revert` ou `invert`? Infelizmente, para string não tem. Mas se eu conheço bem o Java, ou se eu procuro no GUJ (http://www.guj.com.br/) "como inverter uma string em Java". Você encontrará posts como esse (http://www.guj.com.br/t/apresentar-o-valor-da-string-invertida/54643/5) que mencionam o `StringBuffer`. Ao pesquisar mais a fundo (https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html), vemos que o `StringBuffer` tem o método `reverse()`. Em uma maratona, não poderíamos fazer essa pesquisa por não termos acesso a APIs. Nesses casos, conhecer a sua API faz toda a diferença. Vamos aplicar o que aprendemos.

```
private static boolean ehPalindromo(String palavra) {
  new StringBuilder(palavra).reverse()

    //System.out.println(palavra + " " + palavra.lenght());
    for(int i = 0; i < palavra.length() / 2; i++){
      char esquerda = palavra.charAt(i);
      char direita = palavra.charAt(palavra.lenght() - 1 -i);
      //System.out.println(esquerda + " " + direita);
      if(esquerda =! direita){
          return false;
      }

    }
    return true;
```

Esse `StringBuilder` nos retorna outro `StringBuilder`, que podemos transformar em string normal, que chamaremos de `invertido`.

```
private static boolean ehPalindromo(String palavra) {
  String invertido = new StringBuilder(palavra).reverse().toString;

    //System.out.println(palavra + " " + palavra.lenght());
```

```
        for(int i = 0; i < palavra.length() / 2; i++){
          char esquerda = palavra.charAt(i);
          char direita = palavra.charAt(palavra.lenght() - 1 -i);
          //System.out.println(esquerda + " " + direita);
          if(esquerda =! direita){
              return false;
          }

        }
        return true;
```

A partir disso, já podemos apagar todo o restante desse bloco e reescrever o código. Se `palavra` for igual a `invertido`, a palavra é um palíndromo.

```
    private static boolean ehPalindromo(String palavra) {
      String invertido = new StringBuilder(palavra).reverse().toString;
      return palavra.equals(invertido);
```

Vamos testar no terminal?

```
Alura-Azul:bin alura$ pwd
/Users/alura/guilherme/Documents/workspace/palincod/bin
Alura-Azul:bin alura$ wc ../src/Main.java
      44      108     859 ../src/Main.java
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
```

Ainda está funcionando. Mas quantos caracteres esse código tem?

```
Alura-Azul:bin alura$ pwd
/Users/alura/guilherme/Documents/workspace/palincod/bin
Alura-Azul:bin alura$ wc ../src/Main.java
      44      108     859 ../src/Main.java
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
Alura-Azul:bin alura$ wc ../src/Main.java
      34      64     600 ../src/Main.java
```

Conseguimos ir de `859` para `600`. É bastante, mas ainda não o suficiente. A partir daqui não há segredo; é só otimizar o código para que ele tenha apenas `250` caracteres. Até aqui, fizemos melhorias reais no programa, que facilitam a manutenção para a próxima pessoa que lidar com o algoritmo, um grande benefício para programadores.

O que faremos agora não é ganho, pois apertaremos o código ao máximo, dificultando sua compreensão. Não faça isso no seu cotidiano, pois fará todo mundo sofrer ao ler o seu código. Só faça isso em desafios de maratona que peçam especificamente que você digite o minimo possível. Lembre-se: você não é pago no seu trabalho para digitar o mínimo possível. Você é pago para criar um sistema cuja manutenção não seja muito cara. O que faremos agora tornará a

manutenção a mais cara do mundo. Ou seja: não podemos fazer isso no nosso dia a dia como programadores profissionais.