01

Conhecendo o problema dos palíndromos

Transcrição

Pronto para o nosso próximo problema? É o Palincod – Palindrome or not (http://www.spoj.com/problems/PALINCOD/).

> **PALINCOD - Palindrome or not** no tags A palindrome is a word, phrase, number, or other sequence of units that can be read the same way in either direction, with general allowances for adjustments to punctuatiom and word dividers
>
> for eg., HELLOLLEH is a Palindrome string,
>
> and 123456789 is not a Palindrome number.
>
> Input
>
> t - number of testcases [t < 1000].
>
> On each of the next t lines given string.
>
> Output
>
> For each next t lines output the dataset number as a decimal integer (start counting at one), a space and "YES" if the given input is palindrome, or "NO" if the input is not.
>
> Example
>
> ```
> Input:
> 2
> HEllOolLEH
> ILOVEyou
>
> Output:
> 1 "YES"
> 2 "NO"
> ```
>
> CHALLENGE:- minimum size of the code

O enunciado começa definindo o que é palíndromo, uma palavra, frase, número ou outra sequência de unidades (ou seja, se for número contam os dígitos e se for uma palavra contam as letras) que pode ser lida da mesma maneira nas duas direções, da esquerda para a direita ou da direita para a esquerda. E ele menciona regras gerais para ajustes de pontuação e de divisão de palavra, o que soa meio estranho. O exemplo dado não esclarece essa questão. Se fosse um cliente humano, eu prontamente perguntaria o que fazer no caso de ter um hífen ou um ponto. Considerar ou ignorar? Como não é o caso e não está exemplificado, estou pensando que o problema não quer que eu me preocupe com isso.

Analisando a entrada, vemos que haverá menos que `1000` casos. Ou seja, pode haver `0` casos até `999` casos. Seria bizarro ter `-1` caso, e meio inútil ter `0` casos. São dois casos que nós consideramos meio inválidos, mas que o input

considera. O enunciado nos avisa que cada uma dessas linhas `t` é uma string.

E no output descobrimos que cada uma dessas linhas deve estar na saída com um número, que começa a contar em `1`, seguido de um espaço e depois `"YES"` ou `"NO"`, entre aspas, indicando se a string é ou não um palíndromo. Em seguida, mostra-se um exemplo com uma string que é um palíndromo e uma que não é.

Há diversas perguntas que podemos fazer a partir desse enunciado, e teremos que extrapolar a partir delas, pensando no que está explícito, implícito e no que não é citado e teremos que chutar. Vamos tentar implementar um código?

Primeiro, temos que criar um programa. Antes disso, vamos dar uma olhada no nome que o Spoj espera que o programa tenha. Para isso, clicamos em `Submit solution` e vemos o código que já está no campo.

```
import java.util.*;
import java.lang.*;

class Main {
{
    public static void main (String[] args) throws java.lang.Exception
    {

    }
}
```

O nome do programa deve ser `Main`. Criaremos então um novo `Java Project`, de nome `palincod`, e dentro dele, a classe `Main` sem pacote. Começaremos escrevendo o método `main`.

```
public class Main {

    public static void main (String [] args) {

    }
}
```

Agora precisamos criar um código que leia um determinado número de casos, que vêm na primeira linha do arquivo a ser lido. Mas, antes de criar o código, vamos criar os testes, a começar pelo exemplo dado pelo enunciado.

Na maratona, talvez não tenhamos o tempo ou a calma para criar um caso de teste de unidade. Mas já podemos criar um teste com o exemplo dado pelos juízes, afinal, sabemos que se o nosso programa não conseguir lidar com esse exemplo, ele não passará. Assim, nossa `entrada1.txt` será:

```
2
HELloolLEH
ILOVEyou
```

Para ler o número de entradas, criaremos um `Scanner`, algo que fazemos muitas vezes. Geralmente nas maratonas abrevia-se a variável para `sc`, mas como aqui temos tempo, usaremos `scanner` mesmo. O argumento será `System.in`.

```
public class Main {

    public static void main (String [] args) {
```

```
        Scanner scanner = new Scanner(System.in);
    }
}
```

O que esse `Scanner` lerá? Um número de casos, que consideraremos inteiro. Embora isso não esteja explicitado no enunciado, seria um pouco maldoso do cliente usar um número quebrado. Assim, consideraremos que é um `int`. Chamaremos o `t` do enunciado de `casos`.

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
    }
}
```

Agora temos que fazer um `for()` do `1` até o total de casos.

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i < casos; i++){

        }
    }
}
```

Para cada uma das palavras, precisamos imprimir o `i`. Comecei no `1` de propósito. Ao fazer o `for` antigo, no estilo C, é muito comum usarmos o `for(int i = 0)`. Aqui, não estamos varrendo um array, estamos apenas imprimindo qual é o caso atual, que começa no `1`. Para imprimir, um `sysout`, só para ver funcionar. Repare que fomos rapidamente para um teste. Por que testamos rápido? É importante vermos que a leitura e a saída estão corretas, independentemente do algoritmo. Veremos só a saída inicial, mas se não a testarmos e formos completando o código, ficará muito mais difícil encontrar um erro depois.

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i < casos; i++){
            System.out.println(i);

        }
    }
}
```

Vamos então ao terminal. Primeiro, é preciso entrar na pasta certa.

```
Alura-Azul:bin alura$ pwd
/Users/alura/guilherme/Documents/workspace/test/bin
Alura-Azul:bin alura$ cd..
Alura-Azul:test alura$ cd..
Alura-Azul:workspace alura$ pwd
/Users/alura/guilherme/Documents/workspace
Alura-Azul:workspace alura$ ls
RemoteSystemTempFiles Palincod
Alura-Azul:workspace alura$ cd palincod/
Alura-Azul:palincod alura$ cd bin
```

Aqui, ele deve nos mostrar os arquivos do projeto.

```
Alura-Azul:bin alura$ pwd
/Users/alura/guilherme/Documents/workspace/test/bin
Alura-Azul:bin alura$ cd..
Alura-Azul:test alura$ cd..
Alura-Azul:workspace alura$ pwd
/Users/alura/guilherme/Documents/workspace
Alura-Azul:workspace alura$ ls
RemoteSystemTempFiles Palincod
Alura-Azul:workspace alura$ cd palincod/
Alura-Azul:palincod alura$ cd bin
Alura-Azul:bin alura$ ls
Main.class    entrada1.txt
Alura-Azul:bin alura$ java Main < entrada1.txt
```

Encontramos o arquivo de teste. Podemos limpar a tela e rodar.

```
Alura-Azul:bin alura$ java Main < entrada1.txt
1
```

Ele imprimiu só para o primeiro caso. Por que não imprimiu o segundo? Veja, já erramos nesse trecho curtinho de código. Vamos olhá-lo com mais atenção.

```
public class Main {
    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i < casos; i++){
            System.out.println(i);

        }
    }
}
```

Observemos `for(int i = 1; i < casos; i++)`. No caso 2, `i < casos` não é verdadeiro, pois nesse caso `i` é `2` e `casos` é `2`. E `2` não é menor que `2`. Precisaríamos ter colocado um `=` para a inequação ser verdadeira. Assim:

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i <= casos; i++){
            System.out.println(i);

        }
    }
}
```

É preciso ter cuidado, pois há casos em que o número será menor, outros em que será igual. Portanto, manteremos os dois símbolos. Vamos rodar novamente no terminal?

```
Alura-Azul:bin alura$ java Main < entrada1.txt
1
Alura-Azul:bin alura$ java Main < entrada1.txt
1
2
```

Funcionou! Outra possibilidade que substituiria o `<=` seria escrever `for(int i = 1; i < casos+1; i++)`. Vamos testar essa também?

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i < casos+1; i++){
            System.out.println(i);

        }
    }
}
```

No terminal:

```
Alura-Azul:bin alura$ java Main < entrada1.txt
1
Alura-Azul:bin alura$ java Main < entrada1.txt
1
2
Alura-Azul:bin alura$ java Main < entrada1.txt
1
2
```

Como nesse caso queremos do `1` ao `2`, inclusive o `2`, e não do `1` até o `3` (`caso+1`), excluindo o três – o que até soa estranho de se dizer. O que parece mais fácil de não errar? A primeira forma. Assim, usaremos a linha `for(int i = 1; i <= casos; i++)`.

Por isso digo que é tão importante pensar na condição de início. Não importa se você já escreveu `for()` mais de mil vezes. Assim que você colocar os parênteses, deve parar e pensar na condição inicial. Já sabemos que a condição inicial é `1` por não ser um array, mas uma lista de casos. E a condição final? Precisamos chegar e imprimir o caso 2, que é o último, mas deve estar incluso – por isso usamos `>=` . Mas foi preciso pensar se era melhor colocar o `+1` ou não.

Se tivéssemos parado para pensar com muito cuidado no laço, ele já teria saído assim inicialmente. Mas o que acontece geralmente é que ficamos com preguicite aguda de pensar direito e escrevemos o laço de qualquer jeito, esquecendo os detalhes mais importantes como o `<=` .

Para evitar esse tipo de problema, escreva seu teste antes. Assim, você não se esquecerá de nenhum dos casos. Além disso, também é preciso pensar muito bem nas condições do `for` . É muito mais eficiente parar para pensar alguns segundos aqui do que ficar caçando erros quando o código estiver enorme.

Bom, já conseguimos imprimir uma parte do que o problema pede: os números. Mas ele também pede para imprimirmos um espaço, e depois `"YES"` ou `"NO"` , de acordo com a leitura da string. O espaço é fácil de adicionar:

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i < casos+1; i++){
            System.out.println(i + " ");

        }
    }
}
```

Agora precisamos adicionar a palavra `"YES"` ou `"NO"` , com as aspas, de acordo com a palavra. Então, precisamos imprimir a palavra. Para isso, pediremos para o `Scanner` ler a próxima linha, e chamaremos a string de `palavra` .

```
public class Main {

    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        for(int i = 1; i < casos+1; i++){
            System.out.println(i + " ");
            String palavra = scanner.nextLine();

        }
    }
}
```

Já sabemos que há uma palavra. A questão é se ela é um palíndromo ou não. Agora podemos testar o código ou ir para o próximo passo. Só escrevemos uma linha, certo? Colocamos um espaço no `sysout` e lemos a palavra. Por que testar? Estamos sendo confiantes demais. Vamos ao terminal testar.

```
Alura-Azul:bin alura$ java Main < entrada1.txt
1
2
```

Funcionou. Estamos garantidos? Será que tem algo errado? Vá pensando na questão. Até a próxima!