03

Calculando o 3n mais 1

Transcrição

Vamos para o algoritmo. O primeiro número a ser lido nas nossas entradas é o `1`, então vamos começar por ele no algoritmo. Se não formos capazes de calcular para o `1`, não conseguiremos fazer nada.

Vamos fazer em separado, antes mesmo do programa, um `calculaPara(1)`.

```
public class Main {

  public static void main(String[] args){

    calculaPara(1);



      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){
        int i = scanner.nextInt();
        int j = scanner.nextInt();
        System.out.println(i + " " + j + " xxxxxxx");
      }

  }
}
```

Selecionando e pressionando `Ctrl + 1`, cria-se um método, que aparecerá lá embaixo.

```
public class Main {

  public static void main(String[] args){

    calculaPara(1);


      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){
        int i = scanner.nextInt();
        int j = scanner.nextInt();
        System.out.println(i + " " + j + " xxxxxxx");
      }

  }

  private static void calculaPara(int i){
    //TODO Auto-generated method stub
  }
}
```

Vamos deletar o comentário e pensar sobre o algoritmo. Primeiro, temos que imprimir `n`. Se ele for igual a `1`, podemos parar o algoritmo (`return`). Se `n` for ímpar, ou seja, se o resto (`%`) da sua divisão por `2` for `1`, ele deve ser multiplicado por `3` e somar `1`.

```
public class Main {

  public static void main(String[] args){

    calculaPara(1);


      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){
        int i = scanner.nextInt();
        int j = scanner.nextInt();
        System.out.println(i + " " + j + " xxxxxxx");
      }

  }

  private static void calculaPara(int i){
    System.out.println(n);
    if(n == 1) return;
    if(n % 2 == 1) n = n * 3 + 1;
  }
}
```

No caso de `n` ser par, ele deve ser dividido por `2`. E ele deve imprimir o número gerado.

```
public class Main {

  public static void main(String[] args){

    calculaPara(1);


      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){
        int i = scanner.nextInt();
        int j = scanner.nextInt();
        System.out.println(i + " " + j + " xxxxxxx");
      }

  }

  private static void calculaPara(int i){
    System.out.println(n);
    if(n == 1) return;
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    System.out.println(n);
  }
}
```

Vamos rever se é isso mesmo que o enunciado está pedindo?

> Consider the following algorithm: 1. input n 2. print n 3. if n = 1 then STOP 4. if n is odd then n ←− 3n + 1 5. else n ←− n/2 6. GOTO 2

Faltou apenas o sexto passo, que pede um *loop*. Podemos usar o `while(true)` então, certo? Qual é o problema desse `while(true)` ? Você poderia me dizer que tem um `return` ali no meio. Mas não sabemos se para todos os `n` s esse algoritmo termina, embora dentro do intervalo determinado saibamos que sempre terminará. Já conversamos que não devemos usar essa construção, mesmo que deixe o nosso código um pouquinho mais feio. Prefiro deixar um código menos bonito, do que um que esteja sujeito a uma mudança de vírgula de outro programador que o faça entrar em *loop* infinito. Assim, faremos uma condição direta para esse `while()` , que será `n == 1` para parar.

```
private static void calculaPara(int i){
  System.out.println(n);
  while(n == 1); {
    if(n == 1) return;
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    System.out.println(n);
  }
}
```

Vamos testar?

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1
4
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

Ele rodou o algoritmo, imprimiu um `1` e depois um `4` . Tem algo de errado. Nós colocamos como condição `while(n == 1)` , quando na verdade deveria ser `while(n != 1)` . Quando `n == 1` , o algoritmo deve parar.

```
private static void calculaPara(int i){
  System.out.println(n);
  while(n != 1) {
    if(n == 1) return;
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    System.out.println(n);
  }
}
```

Podemos testar novamente.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

Agora ele fez o `1` e acabou, afinal a regra é que ele pare ao encontrar esse número. Nem tem graça. Vamos testar outro, um que o próprio enunciado nos sugere: `22`.

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

  calculaPara(22);


    Scanner scanner = new Scanner(System.in);
    while(scanner.hasNextLine()){
      int i = scanner.nextInt();
      int j = scanner.nextInt();
      System.out.println(i + " " + j + " xxxxxxx");
    }

}

private static void calculaPara(int i){
  System.out.println(n);
  while(n != 1) {
    if(n == 1) return;
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    System.out.println(n);
  }
}
```

No terminal:

```
Alura-Azul:bin alura$ java Main < entrada2.txt
22
11
34
17
52
26
13
40
```

```
20
10
5
16
8
4
2
1
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

A sequência que ele nos mostra é exatamente a mesma do enunciado. Era o que queríamos, nosso algoritmo funciona, ao menos para esses números. Mas a gente realmente quer imprimir esses números? O que queremos é saber quantos números seriam impressos. Começaremos nosso algoritmo com:

```
int impressos = 0;
impressos++;
```

Isso não é a mesma coisa que fazer `int impressos = 1` ? Fica muito mais direto. Esse `impressos++` deve ser colocado ao final do algoritmo, pois toda vez que o número passa por ele, gera mais um. Assim:

```
private static void calculaPara(int i){
  int impressos = 1;
  //System.out.println(n);
  while(n != 1) {
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    impressos++
    //System.out.println(n);
  }

}
```

No final, devemos imprimir o total de números que seriam impressos.

```
private static void calculaPara(int i){
  int impressos = 1;
  //System.out.println(n);
  while(n != 1) {
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    impressos++
    //System.out.println(n);
  }
```

```
  System.out.println("Total " + impressos);
}
```

Agora podemos ver se no caso do `22` realmente estaríamos imprimindo (ou seja, o *cicle lenght*) `16` números, como o enunciado nos diz.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
Total 16
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

Ele imprimiu `16` , como o esperado. Podemos ignorar o restante da saida por enquanto. Para o número `1` ainda está funcionando? Vamos descobrir.

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

  calculaPara(1);
  calculaPara(22);


    Scanner scanner = new Scanner(System.in);
    while(scanner.hasNextLine()){
      int i = scanner.nextInt();
      int j = scanner.nextInt();
      System.out.println(i + " " + j + " xxxxxxx");
    }

}

private static void calculaPara(int i){
  int impressos = 1;
  //System.out.println(n);
  while(n != 1) {
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    impressos++
    //System.out.println(n);
  }
  System.out.println("Total " + impressos);
}
```

E, de novo, vamos para o terminal.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
Total 1
```

```
Total 16
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

Note que, na maratona, os arquivos de teste são os arquivo de entrada de verdade. Em um teste de unidade de programação do cotidiano, como você pode ver nos nossos [cursos de teste de unidade (https://cursos.alura.com.br/course/tdd)](), saberá que criaríamos testes de unidade para `calculaPara()`, por mais simples que seja. Podemos agora testar um caso mais extremo para essa função: o número `999999`.

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

  calculaPara(1);
  calculaPara(22);
  calculaPara(999999);


    Scanner scanner = new Scanner(System.in);
    while(scanner.hasNextLine()){
      int i = scanner.nextInt();
      int j = scanner.nextInt();
      System.out.println(i + " " + j + " xxxxxxx");
    }

}

private static void calculaPara(int i){
  int impressos = 1;
  //System.out.println(n);
  while(n != 1) {
    if(n % 2 == 1) n = n * 3 + 1;
    else n = n / 2;
    impressos++
    //System.out.println(n);
  }
  System.out.println("Total " + impressos);
}

}
```

Vamos ver no terminal o que acontece quando rodamos?

```
Alura-Azul:bin alura$ java Main < entrada2.txt
Total 1
Total 16
Total 259
1 10 xxxxxxx
```

```
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

O programa rapidamente imprimiu `Total 259`. Vamos verificar quanto tempo ele demorou para fazer esse cálculo?

```
Alura-Azul:bin alura$ time java Main < entrada2.txt
Total 1
Total 16
Total 259
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx

real    0m0.137s
user    0m0.125s
sys     0m0.030s
```

O que consideramos é o `user`, o usuário. Foram `0.125s` para fazer a leitura, passar os números pelo algoritmo e cuspir a saída. É bem rápido, pelo menos para esses números.

Na prática, não queremos fazer os `calculaPara()` individuais. Precisamos calcular para todos os números entre `i` e `j`. E também não nos interessa esse `Total` que estamos imprimindo. Queremos o maior dentre os totais de um intervalo de inteiros. Vamos resolver isso em breve. Até lá!