04

Resolvendo 3n mais 1

Transcrição

Vamos para o último problema? É o 3n + 1, do UVa. Vamos começar criando um novo projeto em Java, que se chamará `maraton-3nmais1`. A seguir, a classe `Main` e o método `main`.

Já podemos criar também a `entrada.txt`. Como temos acesso ao enunciado digitalmente, podemos fazer um *copy paste* do input que o cliente nos dá. Mas na vida real quase nunca é assim. O cliente geralmente fala conosco, e nós somos os responsáveis por anotar e digitar, e é preciso ter muito cuidado nesses momentos. Temos que digitar exatamente o que o cliente pediu, cuidando para que não haja espaços a mais ou a menos. Nesse caso, nossa entrada será:

```
1 10
100 200
201 210
900 1000
```

Devemos ler o enunciado agora, não é?

> Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.
>
> 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
```

É o algoritmo que já conhecemos, com os mesmos exemplos. O enunciado nos diz que não é sabido se o algoritmo funciona para todas as entradas possíveis, mas que funciona no intervalo aberto `0 < n < 1000000`. Assim, não precisamos nos preocupar com os casos impossíveis. Lembrando que mesmo em maratonas às vezes temos casos impossíveis.

O enunciado nos explica que *cycle lenght* é a quantidade de números impressos no algoritmo para um determinado número. E que precisamos imprimir o máximo *cycle lenght* presente entre os números `i e j`, mas, a princípio não explicita se é inclusive ou exclusive. Sabemos apenas que o input será de um par de inteiros por linha e que eles estarão no intervalo em que sabemos que o algoritmo funciona bem. Mais adiante o enunciado reforça:

> For any two numbers i and j you are to determine the maximum cycle length over all numbers between and including both i and j.

Agora sabemos que o `i` e o `j` estão **inclusos** no intervalo.

O output deve apresentar `i`, `j` **na mesma ordem** em que apareceram no input, e o maior comprimento de ciclo entre eles, separados por pelo menos um espaço. Devemos imprimir uma linha de output para cada uma do input. O próprio enunciado já nos dá a deixa de que o input pode vir em uma ordem não tão interessante.

Vamos ler o input então. Começaremos com um `Scanner`, usando o `hasNext`. Enquanto houver elementos para ler, o `Scanner` vai funcionar.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while(sc.hasNext()) {

    }

  }
}
```

Mesmo detestando o nome, usaremos `i` e `j` para nos referir aos números porque exatamente assim que o enunciado os chama.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while(sc.hasNext()) {
      int i = sc.nextInt();
      int j = sc.nextInt();
    }

  }
}
```

Agora faremos um `for()` entre esses dois números. Gosto de usar `atual` para ficar claro qual número estamos testando. Ele irá do `i` até o `j`.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while(sc.hasNext()) {
      int i = sc.nextInt();
      int j = sc.nextInt();

      for(int atual = i; atual <= j; atual++){
    }

  }
}
```

Espere um momento. Como temos certeza de que `atual++` não vai estourar o valor de `j`? Quando começamos no `0` e vamos para um número positivo, que é o comum de um array, esse número não estoura. Mas quando usamos o `++` sem

começar no `0`, o `i` já pode começar maior que o `j`. E isso é um problema. Considere como regra geral para todos os `for()` que você fizer na vida: se não começar em `0` e tiver um `++`, a atenção deve ser redobrada. E, se em vez de um `++` tiver um `--`, tome mais cuidado ainda, porque há grandes chances de programarmos errado.

E quais cuidados podemos tomar nesse caso? Não sabemos qual número entre o `i` e o `j` será maior, e essa ordem pode variar linha a linha. Para garantir que sempre pegaremos o menor, usamos o `Math.min(i, j)`. Para o maior `Math.max(i, j)`.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while(sc.hasNext()) {
      int i = sc.nextInt();
      int j = sc.nextInt();

      for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
    }

  }
}
```

Dessa maneira temos certeza de que não teremos problemas. Agora queremos calcular o maior *cicle lenght*, e para isso sabemos que se o maior ciclo até então for o resultado que estamos calculando, ele deve se tornar o `maiorCicloAteEntao`. Definiremos inicialmente o `maiorCicloAteEntao` como `1`, pois é o menor ciclo possível dentro do intervalo, que se inicia em `1`.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while(sc.hasNext()) {
      int i = sc.nextInt();
      int j = sc.nextInt();

      int maiorCicloAteEntao = 1;
      for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
        int resultado = calculaPara(atual);
        if(resultado > maiorCicloAteEntao)
        resultado = maiorCicloAteEntao;
    }

  }
}
```

Agora só falta o `sysout` e efetivamente fazer o `calculaPara`. O `sysout` é fácil, lembrando que os número precisam ser separados por pelo menos um espaço.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while(sc.hasNext()) {
      int i = sc.nextInt();
      int j = sc.nextInt();
```

```
        int maiorCicloAteEntao = 1;
        for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
          int resultado = calculaPara(atual);
          if(resultado > maiorCicloAteEntao)
          resultado = maiorCicloAteEntao;
      }
      System.out.println(i + " " + j + " " + maiorCicloAteEntao);
    }
  }
```

Criando o `calculaPara`, teremos ao final do código:

```
  private static int calculaPara(int atual){
    //TODO Auto-generated method stub
    return 0;
  }

  ...
  }
```

Vamos deletar o comentário e pensar: o que precisamos calcular mesmo? Sabemos que se (`if()`) o `atual` for `1`, é preciso parar `return`. Mas temos que calcular o total de vezes que o número passa pelo algoritmo.

```
  private static int calculaPara(int atual){
    if(atual==1) return;
    return 0;
  }

  ...
  }
```

O que vai acontecer é um loop, por isso o `while()` é mais adequado que o `if()`. E o que nos interessa é o que acontece quando o número for diferente de `1`. Corrigindo:

```
  private static int calculaPara(int atual){
    while(atual!=1) {

    }
    return 0;
  }

  ...
  }
```

Se o `atual` for divisível por `2` (`2%==0`)l, ou seja, par, devemos dividi-lo por `2`.

```
  private static int calculaPara(int atual){
    while(atual!=1) {
      if(atual%2==0) atual = atual / 2;
    }
```

```
  return 0;
}

...
}
```

Se ele for ímpar (portanto, `else`), precisamos multiplicá-lo por `3` e somar `1`.

```
private static int calculaPara(int atual){
  while(atual!=1) {
    if(atual%2==0) atual = atual / 2s
    else
        atual = atual * 3 + 1;
  }
  return 0;
}

...
}
```

Precisamos calcular o total de operações realizadas, que começa com `1`, e aumenta ao final do ciclo. Assim:

```
private static int calculaPara(int atual){

  int operacoes = 1;
  while(atual!=1) {
    if(atual%2==0) atual = atual / 2s
    else
        atual = atual * 3 + 1;
        operacoes++;
  }
  return operacoes;
}

...
}
```

Será que está pronto? Vamos testar. No terminal, depois de entrar no diretório correto, temos:

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

Erramos alguma coisa, pois todos os comprimentos de ciclo estão dando `1`. Vamos rever o código que escrevemos até agora:

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
    while(sc.hasNext()) {
      int i = sc.nextInt();
      int j = sc.nextInt();

      int maiorCicloAteEntao = 1;
      for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
        int resultado = calculaPara(atual);
        if(resultado > maiorCicloAteEntao)
        resultado = maiorCicloAteEntao;
    }
    System.out.println(i + " " + j + " " + maiorCicloAteEntao);
  }

  private static int calculaPara(int atual){

    int operacoes = 1;
    while(atual!=1) {
      if(atual%2==0) atual = atual / 2s
      else
          atual = atual * 3 + 1;
          operacoes++;
    }
    return operacoes;
  }
}
```

Provavelmente erramos em algum lugar perto do `maiorCicloAteEntao`. Mas, não conseguindo encontrar o problema de cara, fazemos um `sysout`. Na vida real podemos fazer o `sysout` ou debugar, mas como na maratona o tempo é escasso, só debugamos em um caso muito grave, em que não conseguimos entender o que está acontecendo.

Vamos tentar imprimir o `atual`.

```
...
int maiorCicloAteEntao = 1;
for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
  System.out.println(atual);
  int resultado = calculaPara(atual);
  if(resultado > maiorCicloAteEntao)
  resultado = maiorCicloAteEntao;
}
...
```

No terminal, temos:

```
Alura-Azul:bin alura$ java Main < entrada.txt
1
2
3
4
5
6
7
8
9
```

```
10
1 10 1
...
```

Parece que ele está imprimindo número a número direitinho, mas mesmo assim imprime `1` como comprimento de ciclo. Então o erro deve estar em outra linha de código. Vamos olhar novamente.

```
...
int maiorCicloAteEntao = 1;
for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
  int resultado = calculaPara(atual);
  if(resultado > maiorCicloAteEntao)
  resultado = maiorCicloAteEntao;
}
...
```

Escrevemos que o `resultado` deve ser igual ao `maiorCicloAteEntao`, que definimos como `1`! É preciso que a equação seja escrita de outra maneira: `maiorCicloAteEntao = resultado`.

```
...
int maiorCicloAteEntao = 1;
for(int atual = Math.min(i, j); atual <= Math.max(i, j); atual++){
  int resultado = calculaPara(atual);
  if(resultado > maiorCicloAteEntao)
   maiorCicloAteEntao = resultado;
}
...
```

Agora sim! Vamos testar novamente.

```
Alura-Azul:bin alura$ java Main < entrada.txt
1 10 20
100 200 125
201 210 89
900 1000 174
```

Parece igual ao que está no enunciado. Vamos extrapolar para outros casos? Sabemos que pode ser até `999999`, então seria interessante testar esse caso. Vamos acrescentá-lo na `entrada.txt`. Seria bom testar também do `1` até ele mesmo.

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

No terminal:

```
Alura-Azul:bin alura$ java Main < entrada.txt
1 10 20
100 200 125
201 210 89
900 1000 174
999999 999999 259
1 1 1
```

E se a entrada tiver uma quebra de linha no final?

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

Testando no terminal:

```
Alura-Azul:bin alura$ java Main < entrada.txt
1 10 20
100 200 125
201 210 89
900 1000 174
999999 999999 259
1 1 1
```

Tudo parece normal. Vamos testar um caso mais extremo: do `1` até o `999999`.

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

No terminal:

```
Alura-Azul:bin alura$ java Main < entrada.txt
1 10 20
100 200 125
201 210 89
900 1000 174
999999 999999 259
1 1 1
```

Ele travou. Repare que, durante a maratona, se eu caísse nesse exercício, eu melhoraria esse programa. Mesmo que o juiz não tenha pedido, pois a especificação do programa pede. Eu tentaria otimizar o programa para essa situação.

Nesse curso não iremos focar nessa otimização (*fine tuning*), mas podemos abordar esse e outros assuntos de melhorias estruturais de programação em um próximo curso.

Ainda faltou testarmos o caso em que os números vêm em ordem decrescente. Vamos acrescentar na `entrada.txt` e tirar o caso que não está funcionando.

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

Vamos testar novamente:

```
Alura-Azul:bin alura$ java Main < entrada.txt
1 10 20
100 200 125
201 210 89
900 1000 174
999999 999999 259
10 1 20
200 100 125
210 201 89
1000 900 174
1 1 1
```

A ordem dos números permaneceu e o comprimento de ciclo está correto. Podemos copiar todo o nosso código e submeter no site. Após colá-lo e enviá-lo, vamos em `My Submissions` para ver o veredicto.

My Submissions

#	Problem	Verdict	Language
18432818	100 The 3n + 1 problem	Accepted	JAVA
18432699	100 The 3n + 1 problem	Accepted	JAVA

A solução foi aceita!