---

1. **Introdução**

   O objetivo desta prática é construir uma estrutura de dados chamada árvore de Fenwick, que representa uma tabela de inteiros cujos índices variam de 0 a $n-1$ com duas operações:

   - **void increment(int i, int delta, FenwickTree t)** acrescenta delta (positivo ou negativo) ao valor contido no índice i da árvore t,
   - **int prefixSum(int i, FenwickTree t)** que retorna a soma de todos os elementos de t cujos índices variam entre 0 e $i-1$.

   Nós procuramos uma solução em que ambas as operações têm custo $O(\log n)$. A ideia é a utilização de uma árvore de Fenwick: as folhas contêm os elementos da tabela e cada nó interno contém a soma de todas as folhas que se encontram abaixo. Além disso:

   - cada nó interno tem uma sub-árvore esquerda e uma sub-árvore direita **ambas não vazias**,
   - cada nó interno também guarda o número de folhas na sua sub-árvore esquerda (que, portanto, é sempre 0 para uma folha),
   - uma árvore Fenwick é **balanceada**, quer dizer que para qualquer nó x, a altura da subárvore esquerda de x e a altura da sub-árvore direita de x não diferem em mais de 1 unidade.

   A figura 1 mostra uma possível representação de uma árvore de Fenwick para uma tabela de 6 inteiros.

2. **Campos e construtores**

   Escreva um em arquivo FenwickTree.java uma classe pública FenwickTree que representa um nó de uma árvore Fenwick, com os quatro campos a seguir:

   - **int value**, que contém:
     - para uma folha, o valor a que ela se refere
     - para um nó interno, a soma dos valores das folhas abaixo dele;
   - **int leftSize**, que contém:
     - para uma folha, o valor 0
     - para um nó interno, o número de folhas na sub-árvore esquerda;
   - **FenwickTree left** e **FenwickTree right**, que contêm:
     - para uma folha, null
     - para um nó interno, ponteiros para as sub-árvores esquerda e direita, respectivamente.

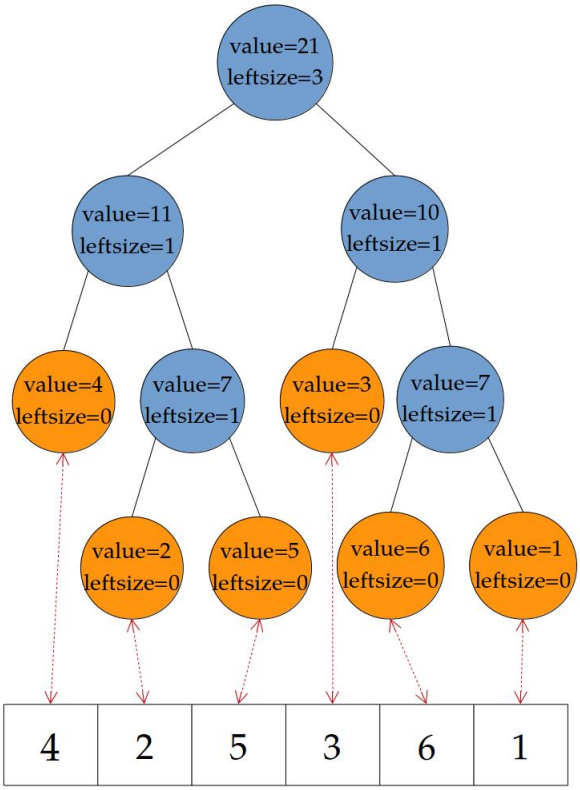

Figure 1: Exemplo de uma Fenwick Tree

Adicione à classe FenwickTree:

- Um construtor **FenwickTree(int value)** que constrói uma folha referente ao valor **value**,
- Um construtor **FenwickTree(int leftSize, FenwickTree left, FenwickTree right)** que constrói um nó interno, dadas duas árvores (de Fenwick) esquerda e direita supostamente não-vazias; assume-se que leftSize é o número de folhas na sub-árvore esquerda e que as alturas das árvores esquerda e direita não diferem de mais de uma unidade.
- Um método **public String toString()** que retorna a cadeia formada pelos conteúdos dos campos [**valor, leftSize, esquerda, direita**] (para os campos esquerda e direita, chamamos o método **toString** de forma recursiva).

Você pode testar seu código utilizando o código a seguir, que constrói a árvore FenwickTree (3) e a árvore da figura 1.

```
public class Test{
  public static void main(String[] args){
    System.out.println("Construcao de FenwickTree(3) : " + new FenwickTree(3));
    System.out.println("Construcao da arvore da figura : " +
    new FenwickTree(3, new FenwickTree(1, new FenwickTree(4),
    new FenwickTree(1, new FenwickTree(2), new FenwickTree(5))),
    new FenwickTree(1, new FenwickTree(3),
    new FenwickTree(1, new FenwickTree(6), new FenwickTree(1)))));
  }
}
```

O resultado deve ser:

```
Construcao de FenwickTree(3) : [3, 0]
Construcao da arvore da figura : [21, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]],
[10, 1, [3, 0], [7, 1, [6, 0], [1, 0]]]]
```

3. **Representação de uma tabela cheia de zeros**

   O método **static FenwickTree allZeros(int n)** constrói uma árvore de Fenwick balanceada, contendo n folhas cujos valores são `0`. Presume-se que $n > 0$. Tente compreendê-lo a fim de saber como a recursão se dá tipicamente uma estrutura de dados do tipo árvore.

```
static FenwickTree allZeros(int n){
    if (n==0) return null;
    if (n==1) return new FenwickTree(0);
    int m = n/2;
    return new FenwickTree(0, n-m, allZeros(n-m), allZeros(m));
}
```

   Teste o código usando com a função **main** abaixo que constrói as árvores **allZeros(3)**, **allZeros(4)**, **allZeros(5)** e **allZeros(6)**.

```
public static void main(String[] args){
  System.out.println("Construcao de allZeros(3) : " + FenwickTree.allZeros(3));
  System.out.println("Construcao de allZeros(4) : " + FenwickTree.allZeros(4));
  System.out.println("Construcao de allZeros(5) : " + FenwickTree.allZeros(5));
  System.out.println("Construcao de allZeros(6) : " + FenwickTree.allZeros(6));
}
```

   O resultado deve ser:

```
Construcao de allZeros(3) : [0, 2, [0, 1, [0, 0], [0, 0]], [0, 0]]
Construcao de allZeros(4) : [0, 2, [0, 1, [0, 0], [0, 0]], [0, 1, [0, 0], [0, 0]]]
Construcao de allZeros(5) : [0, 3, [0, 2, [0, 1, [0, 0], [0, 0]], [0, 0]], [0, 1,
[0, 0], [0, 0]]]
Construcao de allZeros(6) : [0, 3, [0, 2, [0, 1, [0, 0], [0, 0]], [0, 0]], [0, 2,
[0, 1, [0, 0], [0, 0]], [0, 0]]]
```

4. **Tamanho de uma árvore**

   Escreva um método **int size()** que retorna o número de folhas da árvore de Fenwick. Atenção! A função tem que ser executada em tempo $O(\log n)$.

   Você pode testar seu código usando o código abaixo que calcula o tamanho das árvores **FenwickTree(6)**, **allZeros(6)** e **allZeros(12)**.

```
public static void main(String[] args){
  // teste de correcao
  System.out.println("Verificacao de correcao da funcao...");
  System.out.println("Tamanho de FenwickTree(6) :    "
  + (new FenwickTree(6)).size());
  System.out.println("Tamanho de allZeros(6) :       "
  + (FenwickTree.allZeros(6)).size());
  System.out.println("Tamanho de allZeros(12) :      "
  + (FenwickTree.allZeros(12)).size());
  FenwickTree T1 = new FenwickTree(3, new FenwickTree(1, new FenwickTree(4),
  new FenwickTree(1, new FenwickTree(2), new FenwickTree(5))),
  new FenwickTree(1, new FenwickTree(3),
  new FenwickTree(1, new FenwickTree(6), new FenwickTree(1))));
  System.out.println("Árvore this : " + T1);
  System.out.println("Tamanho de this :      " + T1.size());
}
```

   O resultado deve ser:

```
Tamanho de FenwickTree(6) :    1
Tamanho de allZeros(6) :       6
Tamanho de allZeros(12) :      12
Árvore this : [21, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]], [10, 1, [3, 0],
 [7, 1, [6, 0], [1, 0]]]]
Tamanho de this :      6
```

5. **Alterando o valor de uma folha**

   As folhas de uma árvore de Fenwick são numeradas da esquerda para a direita começando do índice 0, isto é, a folha de índice 0 de uma árvore de Fencwick t é a folha mais à esquerda, enquanto a folha mais à direita possui índice $n-1$ se t possui n folhas.

Escreva um método **void increment(int i, int delta)** que altera adiciona *delta* ao valor da folha de índice i da árvore de Fenwick. Supõe-se que $0 \leq i < n$ onde n é o número de folhas da árvore. Garanta que a árvore, após alteração, seja uma árvore Fenwick, ou seja, que as informações nos campos dos nós internos sejam devidamente atualizadas.

Utilize código abaixo que constrói a árvore da figura 1, a partir da estrutura contendo apenas valores 0.

```
public static void main(String[] args){
  // teste de correcao
  System.out.println("Verificacao de correcao da funcao...");
  FenwickTree T = new FenwickTree(3,
  new FenwickTree(1, new FenwickTree(0),
  new FenwickTree(1, new FenwickTree(0), new FenwickTree(0))),
  new FenwickTree(1, new FenwickTree(0),
  new FenwickTree(1, new FenwickTree(0), new FenwickTree(0))));
  System.out.println("Arvore this :                  " + T);
  T.increment(0, 4);
  System.out.println("Resultado de increment(0, 4) : " + T);
  T.increment(1, 2);
  System.out.println("Resultado de increment(1, 2) : " + T);
  T.increment(2, 5);
  System.out.println("Resultado de increment(2, 5) : " + T);
  T.increment(3, 3);
  System.out.println("Resultado de increment(3, 3) : " + T);
  T.increment(4, 6);
  System.out.println("Resultado de increment(4, 6) : " + T);
  T.increment(5, 1);
  System.out.println("Resultado de increment(5, 1) : " + T);
}
```

O resultado deve ser:

```
Arvore this : [0, 3, [0, 1, [0, 0], [0, 1, [0, 0], [0, 0]]],
 [0, 1, [0, 0], [0, 1, [0, 0], [0, 0]]]]
Resultado de increment(0, 4) : [4, 3, [4, 1, [4, 0], [0, 1, [0, 0], [0, 0]]],
 [0, 1, [0, 0], [0, 1, [0, 0], [0, 0]]]]
Resultado de increment(1, 2) : [6, 3, [6, 1, [4, 0], [2, 1, [2, 0], [0, 0]]],
 [0, 1, [0, 0], [0, 1, [0, 0], [0, 0]]]]
Resultado de increment(2, 5) : [11, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]],
 [0, 1, [0, 0], [0, 1, [0, 0], [0, 0]]]]
Resultado de increment(3, 3) : [14, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]],
 [3, 1, [3, 0], [0, 1, [0, 0], [0, 0]]]]
Resultado de increment(4, 6) : [20, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]],
 [9, 1, [3, 0], [6, 1, [6, 0], [0, 0]]]]
```

```
Resultado de increment(5, 1) : [21, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]],
 [10, 1, [3, 0], [7, 1, [6, 0], [1, 0]]]]
```

6. **Soma das primeiras folhas**

   Escreva um método **int prefixSum(int upto)** que retorna a soma dos valores das folhas da árvore de Fenwick com indíces entre 0(incluído) e **upto**(excluído). Supõe-se que $0 \leq$ **upto** $< n$, onde n é o número de folhas da árvore. Assegure que sua função é executada em tempo $O(\log n)$.

   Utilize o código abaixo para teste. Elecalcula as somas das primeiras folhas da árvore da figura 1.

```
public static void main(String[] args){
  // teste de correcao
  System.out.println("Verificacao de correcao da funcao...");
  FenwickTree T = new FenwickTree(3, new FenwickTree(1, new FenwickTree(4),
  new FenwickTree(1, new FenwickTree(2), new FenwickTree(5))),
  new FenwickTree(1, new FenwickTree(3),
  new FenwickTree(1, new FenwickTree(6), new FenwickTree(1))));
  System.out.println("Arvore this : " + T);
  System.out.println("Soma das primeiras folhas : ");
  for(int upto = 0; upto <= 6; upto++)
      System.out.println("prefixSum(" + upto + ") : " + T.prefixSum(upto));
}
```

   O resultado deve ser:

```
Árvore this : [21, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]],
 [10, 1, [3, 0], [7, 1, [6, 0], [1, 0]]]]
Soma das primeiras folhas :
prefixSum(0) : 0
prefixSum(1) : 4
prefixSum(2) : 6
prefixSum(3) : 11
prefixSum(4) : 14
prefixSum(5) : 20
prefixSum(6) : 21
```

7. **Soma de uma gama de folhas**

   Escreva um método **int between(int lo, int hi)** que retorna a soma dos valores das folhas da árvore de Fenwick, compreendidas entre **lo**(incluído) e **hi**(excluído). Suponha que que $0 \leq$ **lo** $\leq$ **hi** $\leq n$, onde n é o número de folhas da árvore.

   Teste a sua função com o código abaixo que calcula as somas de todas as folhas da árvore da figura 1.

```
public static void main(String[] args){
  FenwickTree T = new FenwickTree(3, new FenwickTree(1, new FenwickTree(4),
  new FenwickTree(1, new FenwickTree(2), new FenwickTree(5))),
  new FenwickTree(1, new FenwickTree(3),
  new FenwickTree(1, new FenwickTree(6), new FenwickTree(1))));
  System.out.println("Arvore this : " + T);
  System.out.println("Soma das folhas entre lo e hi : ");
  System.out.print("         ");
  for(int lo = 0; lo <= 6; lo++)
     System.out.print("lo = " + lo + "  ");
  System.out.println();
  for(int hi = 0; hi <= 6; hi++){
      System.out.print("hi = " + hi + "   ");
      for(int lo = 0; lo <= hi; lo++){
         System.out.print(T.between(lo, hi) + "      ");
         if(T.between(lo, hi) < 10) System.out.print(" ");
      }
      System.out.println();
   }
}
```

O resultado deve ser:

```
Árvore this : [21, 3, [11, 1, [4, 0], [7, 1, [2, 0], [5, 0]]], [10, 1, [3, 0],
 [7, 1, [6, 0], [1, 0]]]]
Soma das folhas entre lo e hi :
         lo = 0  lo = 1  lo = 2  lo = 3  lo = 4  lo = 5  lo = 6
hi = 0   0
hi = 1   4       0
hi = 2   6       2       0
hi = 3   11      7       5       0
hi = 4   14      10      8       3       0
hi = 5   20      16      14      9       6       0
hi = 6   21      17      15      10      7       1       0
```

*Este trabalho prático é de autoria de Jean-Christophe Filliâtre (Poly, France)