Каждая подзадача должна решаться рекурсивно. В частности, во всем задании запрещены циклы, и их использование приведет к начислению 0 баллов за соответствующие подзадачи. То же самое относится и к потокам (кто их уже знает).

Разрешены частные вспомогательные методы и (статические) вспомогательные переменные. Однако методы должны работать без дополнительных вызовов (то есть так, как они используются в основном методе). Следует также отметить, что тесты выполняют несколько вызовов методов подряд, а статические вспомогательные переменные не сбрасываются между ними.

Сигнатура указанных методов (имя, тип возвращаемого значения, типы параметров) не должна изменяться.

1. Сначала вам нужно реализовать новую серию. Это зависит от начального вектора

p=(p0, p1, p2) и следует этому правилу расчета:

pinguSequenceRec(0)=p0

pinguSequenceRec(1)=p1

pinguSequenceRec(2)=p2

pinguSequenceRec(n)=2∗pinguSequenceRec(−n), если n<0

pinguSequenceRec(n) = pinguSequenceRec(n−1) − pinguSequenceRec(n−2) + 2∗pinguSequenceRec(n−3) если n>0

Дополнительные условия:

-122<n<145

Вычисление должно завершиться менее чем за секунду для каждого ввода на среднем ПК. Поэтому для большего n (~> 30) вам нужно что-то придумать.

При проверенных комбинациях n и p в результате не должно быть превышений лимита long

Пример для p = (1,1,2)

Task 1 example output

0: 1

1: 1

2: 2

3: 3

4: 3

5: 4

6: 7

7: 9

8: 10

9: 15

1. Затем вам нужно реализовать близнецовые серии pinguF и pinguM, которые рассчитываются следующим образом:

pinguF(0) = 1

pinguF(n) = n − pinguM(pinguF(n−1)), если n>0

pinguM(0) = 0

pinguM(n) = n − pinguF(pinguM(n−1)), если n>0

n всегда положительный и переполнение int не произойдет

1. Результат зависит от двух входных чисел n и m и также может быть вычислен рекурсивно с учетом промежуточного результата. Изначально, конечно, промежуточный результат равен 0.

если n=0, то окончательный результат является суммой m и промежуточного результата.

В противном случае необходимо различать, является ли сумма n и промежуточного результата четной или нет.

Если оно четное, оно продолжается рекурсивно, давая новому n значение m, а новому m — половину старого n (округленного в меньшую сторону). Кроме того, к промежуточному результату добавляются n-half (старые n, округленные в меньшую сторону).

Если указанная выше сумма нечетная, расчет также продолжается рекурсивно. Однако в этом случае к промежуточному результату прибавляется m, n уменьшается на единицу и m делится пополам (округляется в меньшую сторону).

Опять же, в этой задаче используются только неотрицательные входные данные, и переполнения int не произойдет. В этом упражнении «округление» означает нормальное поведение округления в Java (направление 0).

0, 0: 0

0, 1: 1

0, 2: 2

0, 3: 3

0, 4: 4

0, 5: 5

0, 6: 6

0, 7: 7

0, 8: 8

**0, 9: 9**

**----------**

1, 0: 0

1, 1: 1

1, 2: 3

1, 3: 4

1, 4: 6

1, 5: 7

1, 6: 9

1, 7: 10

1, 8: 12

**1, 9: 13**

**----------**

2, 0: 2

2, 1: 1

2, 2: 2

2, 3: 3

2, 4: 4

2, 5: 3

2, 6: 6

2, 7: 8

2, 8: 8

**2, 9: 7**

**----------**

3, 0: 2

3, 1: 1

3, 2: 3

3, 3: 4

3, 4: 6

3, 5: 7

3, 6: 9

3, 7: 11

3, 8: 12

**3, 9: 13**

**----------**

4, 0: 4

4, 1: 5

4, 2: 4

4, 3: 5

4, 4: 6

4, 5: 9

4, 6: 9

4, 7: 9

4, 8: 10

**4, 9: 14**

**----------**

5, 0: 4

5, 1: 3

5, 2: 7

5, 3: 6

5, 4: 8

5, 5: 9

5, 6: 11

5, 7: 11

5, 8: 14

**5, 9: 15**

**----------**

6, 0: 6

6, 1: 5

6, 2: 7

6, 3: 5

6, 4: 7

6, 5: 7

6, 6: 9

6, 7: 10

6, 8: 11

**6, 9: 13**

**----------**

7, 0: 6

7, 1: 5

7, 2: 7

7, 3: 8

7, 4: 11

7, 5: 9

7, 6: 11

7, 7: 13

7, 8: 15

**7, 9: 17**

**----------**

8, 0: 8

8, 1: 10

8, 2: 8

8, 3: 10

8, 4: 10

8, 5: 12

8, 6: 11

8, 7: 15

8, 8: 14

**8, 9: 16**

**----------**

9, 0: 8

9, 1: 7

9, 2: 12

9, 3: 10

9, 4: 12

9, 5: 14

9, 6: 16

9, 7: 15

9, 8: 18

**9, 9: 19**

1. Вычислите DNS. Двоичное представление значений, которое передается вам в виде int, особенно актуально для рекурсивных вычислений.

Чтобы определить строку DNS:

* Если оба f и m значения „f“ и „m“ равны 0, результирующий DNS представляет собой пустую строку String «».
* Если только f равно 0, то к каждому биту числа m, кроме начальных «0» битов, добавляется «A».
* В противоположном случае ( m = 0) , добавляется «T».

Итак, pinguDNA(0,2) возвращает «AA», так как 2=10b. Для pinguDNA(10,0) решением будет «TTTT», поскольку 10=1010b.

Eсли оба значения f и m не равны 0. В этом случае сначала необходимо определить, заканчиваются ли f и m одним и тем же битовым значением (подсказка: четное/нечетное).

Если да, то DNS зависит от того, является ли f>m ("GT"), f<m ("GA") или f=m ("GC").

Если нет, важно, какое значение заканчивается битом 1. Если f заканчивается таким образом, добавляется «TC», в противном случае — «AC».

Каждый бит используется ровно для одного азотистого основания в выходной строке и поэтому может быть отброшен для рекурсии (совет: разделите на 2).

Вы можете найти пошаговый пример, открыв следующий пример.

**pinguDNA**(21, 25) = GCACTCGAGA

21 in binär ist "10101b"   
25 in binär ist "11001b"

21 != 0 != 25, оба заканчиваются на 1 бит и 21 < 25 => "GA"

Удалить биты => f="1010b"=10, m="1100b"=12

f,m != 0, заканчивается битом 0, f "GA" => общее "GAGA"

Удалить биты => f="101b"=5, m="110b"=6

f,m != 0, заканчивается другим битовым значением, m делится на 2 => "TC" => общее "TCGAGA"

Удалить биты => f="10b"=2, m="11b"=3

f,m != 0 заканчиваются другим битовым значением, f делится на 2 => "AC" => общее "ACTCGAGA"

Удалить биты => f="1b"=1, m="1b"=1

f,m != 0, заканчиваются на 1 бит и равны => "GC" => общее "GCACTCGAGA"

Удалить биты => f=0, m=0 => "", done => всего "GCACTCGAGA"