Practical Task 8.2

(High Distinction Task)

Submission deadline: 11:59 pm Sunday, September 20th
Discussion deadline: 11:59 pm Sunday, October 4th

Task Objective

In this task you will use your advanced knowledge and skills developed through this unit to implement a solution to a complex programming problem .

Task Details

Simon, likes special coins, and for any non-negative integer $k = 0, 1, 2, 3, \ldots$ he has exactly two coins of value $n = 2^k$ (i.e., 2 to the power of k). For a given value Z, help Simon to calculate the number of different ways that can represent Z with the coins which he has. Note that two representations are considered different if there is a coin that appears a *different* number of times (or does not appear at all) in the representations.

1. Download the source code attached to this task. Create a new Microsoft Visual Studio project and import the three files. Your newly built project should compile without errors. Note that the file Tester.cs already provides you with a Main method as a starting point of your program and is important for the purpose of debugging and testing. Another file, TestGenerator.cs, serves the Tester class in Tester.cs with hard-coded test instances. Finally, CoinRepresentation.cs contains a template for the algorithm that you need to develop and implement as part of the CoinRepresentation class.

2. Explore the existing CoinRepresentation class to find the static method named Solve. This method must implement your algorithm. It accepts only a single argument: A *target value* Z that you must match exactly with a set of existing coins. The method must tell Simon the number of ways that his coins can represent the given target value. Note that both the *input argument* and the *output* are *long* integers (i.e. 64-bit integers designated via the *long* data type in C#). You may assume that Z will be between 1 and 1,000,000,000,000,000,000 (10^{18}), inclusive

3. Consider the following examples:
   - For a given input value $Z = 1$, the algorithm must return 1. The only possible way for Simon to represent Z in this case is to use exactly one coin of value 1.
   - For a given input value $Z = 6$, the algorithm must return 3. The following three representations are possible for Simon in this case: {1, 1, 2, 2}, {1, 1, 4} and {2, 4}.
   - For a given input value $Z = 47$, the algorithm must return 2.
   - For a given input value Z =256, the algorithm must return 9.
   - For a given input value Z =8489289, the algorithm must return 6853.
   - For a given input value Z =1000000000, the algorithm must return 73411.

4. Remember that you are free to write your program code within the CoinRepresentation.cs as you wish. The only requirement is to meet the expected signature of the Solve method. Therefore, you may add any extra private methods and attributes if necessary.

   As a *hint for your algorithmic solution*, remember that computer memory is another important resource that you have access to, and its use can significantly reduce computational time, especially when runtime is dramatically long. Furthermore, to speed up computations, you should avoid examining infeasible solutions that violate the given constraints or cannot match the given target

value *exactly*. These two remarks should considerably enhance performance of your solution technique.

Try not only to solve the problem, but also achieve runtime similar to what is stated for each of the test instances in the printout below.

5. As you progress with the implementation of your algorithmic solution, you should start using the Tester class in order to test your code for potential logical issues and runtime errors. This (testing) part of the task is as important as coding.
6. Finally, before submitting ensure your code meets a professional standard with full documentation.
   - Professional code will be correctly formatted, see Microsoft's C# coding conventions:

     https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

     Note: your code should not retain any debug code. It should name variables, properties, methods, class etc appropriately. Your code should not utilise any obfuscations and not retain any unreachable code. Correctly break down task into appropriate methods.

   - Professional documentation is also expected, see Microsoft's C# Documentation conventions:

     https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/documentation-comments

     Note: you must include standard comments explaining code inside each method, plus use formal documentation standard in the above link for classes, properties, methods, delegates etc.

     Second Note: A correct solution that does not follow the above standards will only receive a D grade as a maximum. No HD grade will be granted for code that does not meet the above standards.

Expected Printout

Appendix A provides an example printout for your program. If you are getting a different printout then your implementation is not ready for submission. Please ensure your program prints out Success for all tests before submission.

Further Notes

Reading sections 5.1, 5.3, 5.4, and 10.2 of the course book "Data Structures and Algorithms in Java" by Michael T. Goodrich, Irvine Roberto Tamassia, and Michael H. Goldwasser (2014) should help you with the solution to this task. You may access the book on-line for free from the reading list application in CloudDeakin available in Resources → Additional Course Resources → Resources on Algorithms and Data Structures → Course Book: Data structures and algorithms in Java

Marking Process and Discussion

Same as task 4.2, but unlike other tasks, this task will be graded out of 3. If the task is not at sufficient standard to be marked as complete, then it will be listed as "Fail" or "Exceeded feedback". If it is accepted as complete, then it will receive one of 3 grades as follows:

1. Correct solution but not efficient and/or does not meet coding and documentation standards – Can be counted as D task.
2. Correct solution that meet coding and documentation standards - HD
3. Correct and efficient solution (substantially faster than our solution when run on our computer) that meets coding and documentation standards – High HD

Your tutor will indicate after your submission the level the submission is being considered at. If this is "resubmit" or a score of 1 then they may provide advice on how to improve your submission. You can only have one resubmit to improve your work. To achieve that mark though you will need to show a sufficiently high quality interview.

A result of HD or D will be treated as other HD or D tasks. A High HD will result in a bonus of 5% to your final unit grade. The High HD grade is not expected to be given out to anyone. If it is awarded it will be due to an exceptional submission agreed to by all tutors.

Criteria being considered when marking:

- Correct Solution is a must for all grades including D.
- Professional
    - Use professional coding conventions.
    - Extensive in method commenting.
    - Have full and complete professional documentation.
    - Referenced any materials utilised in solving the task.
- Illustrated understanding of base algorithms used
- Algorithm space/time complexity is provided and explained in the documentation.
- Interview showing clear understanding.
- Illustration of self-directed learning.

Expected Printout

This section displays the printout produced by the attached Tester class, specifically by its *Main* method. It is based on our solution. The printout is provided here to help with testing your code for potential logical errors. It demonstrates the correct logic rather than an expected printout in terms of text and alignment.

```
Attempting test instance 0 with 1 as the argument and 1 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0013283)

Attempting test instance 1 with 6 as the argument and 3 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0002380)

Attempting test instance 2 with 47 as the argument and 2 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000550)

Attempting test instance 3 with 256 as the argument and 9 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000218)

Attempting test instance 4 with 8489289 as the argument and 6853 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000588)

Attempting test instance 5 with 1000000000 as the argument and 73411 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000317)

Attempting test instance 6 with 100 as the argument and 19 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000126)

Attempting test instance 7 with 128 as the argument and 8 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000063)

Attempting test instance 8 with 1073741824 as the argument and 31 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000624)

Attempting test instance 9 with 6370 as the argument and 175 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000165)

Attempting test instance 10 with 10 as the argument and 5 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000162)

Attempting test instance 11 with 2 as the argument and 2 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000038)

Attempting test instance 12 with 3 as the argument and 1 as the expected answer
 :: SUCCESS (Time elapsed 00:00:00.0000035)
```

Attempting test instance 13 with 4 as the argument and 3 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000042)

Attempting test instance 14 with 2000000000 as the argument and 81034 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000366)

Attempting test instance 15 with 999999999 as the argument and 7623 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000669)

Attempting test instance 16 with 1000000000000000000 as the argument and 554817437 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001276)

Attempting test instance 17 with 576460752303423488 as the argument and 60 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002782)

Attempting test instance 18 with 640 as the argument and 23 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000292)

Attempting test instance 19 with 785 as the argument and 34 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000267)

Attempting test instance 20 with 1022 as the argument and 10 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000296)

Attempting test instance 21 with 962 as the argument and 38 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000296)

Attempting test instance 22 with 640 as the argument and 23 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000990)

Attempting test instance 23 with 1099510542205 as the argument and 17863 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000832)

Attempting test instance 24 with 944875173846 as the argument and 1243789 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002454)

Attempting test instance 25 with 672031828383 as the argument and 500073 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0003106)

Attempting test instance 26 with 893915235088 as the argument and 243779 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000966)

Attempting test instance 27 with 1088385987371 as the argument and 4634234 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000684)

Attempting test instance 28 with 347905064087584832 as the argument and 5150282 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001438)

Attempting test instance 29 with 309341003709448449 as the argument and 19102955 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001138)

Attempting test instance 30 with 263810380166378775 as the argument and 4693345949 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002154)

Attempting test instance 31 with 361431780114432130 as the argument and 94727263 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001696)

Attempting test instance 32 with 378311177695920400 as the argument and 20702253 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001893)

Attempting test instance 33 with 290553370434404484 as the argument and 146293655 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000959)

Attempting test instance 34 with 423901414250789313 as the argument and 292614203 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002351)

Attempting test instance 35 with 438190581230404958 as the argument and 6012372582 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002038)

Attempting test instance 36 with 293666568548731467 as the argument and 3648043185 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002884)

Attempting test instance 37 with 392393882169705920 as the argument and 3341296806 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001135)

Attempting test instance 38 with 376370659955075108 as the argument and 3279511256 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001636)

Attempting test instance 39 with 412658913555584867 as the argument and 3498747798 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001117)

Attempting test instance 40 with 999999999999999999 as the argument and 29665503 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001329)

Attempting test instance 41 with 410054521552536292 as the argument and 26030230909 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000878)

Attempting test instance 42 with 416860608518791589 as the argument and 8015276820 as the expected answer

:: SUCCESS (Time elapsed 00:00:00.0001667)

Attempting test instance 43 with 393014244375683364 as the argument and 16905456859 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001646)

Attempting test instance 44 with 518010418436963490 as the argument and 15340957057 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001191)

Attempting test instance 45 with 576460730781662959 as the argument and 794365 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000719)

Attempting test instance 46 with 565764701561028461 as the argument and 2186952 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001096)

Attempting test instance 47 with 571954850028252927 as the argument and 7287457 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002598)

Attempting test instance 48 with 558161296277634687 as the argument and 1416255 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001209)

Attempting test instance 49 with 504314853196816127 as the argument and 6183662 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001258)

Attempting test instance 50 with 123456789 as the argument and 51639 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001618)

Attempting test instance 51 with 768614336404564650 as the argument and 2504730781961 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0002147)

Attempting test instance 52 with 384307168202282325 as the argument and 956722026041 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001805)

Attempting test instance 53 with 384307168202282324 as the argument and 1548008755920 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001043)

Attempting test instance 54 with 192153584101141162 as the argument and 956722026041 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001181)

Attempting test instance 55 with 196657183728511722 as the argument and 502131822759 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001664)

Attempting test instance 56 with 193349852752161450 as the argument and 484936992181 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001611)

Attempting test instance 57 with 196731950519200426 as the argument and 350312970581 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001156)

Attempting test instance 58 with 192153584101141166 as the argument and 644603021052 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001346)

Attempting test instance 59 with 10000000000000000 as the argument and 17165857 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001562)

Attempting test instance 60 with 200 as the argument and 26 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0000130)

Attempting test instance 61 with 93459834598323452 as the argument and 317400926 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0001241)

Attempting test instance 62 with 1717161617181871 as the argument and 69493195 as the expected answer
:: SUCCESS (Time elapsed 00:00:00.0003720)

Summary: 63 tests out of 63 passed

Tests passed (0 to 63): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62

Tests failed (0 to 63): none