Module 2 Critical Thinking: Algorithm Analysis

Donghwan Kim

Colorado State University Global

CSC 506: Design and Analysis of Algorithms

Dr. Lori Farr

May 4, 2022

Module 2 Critical Thinking: Algorithm Analysis

This assignment is concerned with the task of calculating and printing the daily salary of a worker who each day is paid twice the previous day's salary. It develops algorithms based on loop structure and recursive structure and also discusses how the numbers appearing in the algorithms, integer and real number, are stored. The computational problem of a worker whose salary is doubled daily can be expressed as the recurrence relation of $\text{salary}_d = 2 * \text{salary}_{d-1},\ d = 1, 2, \cdots$ with the initial condition is given by $\text{salary}_0 = 0.01$. Algorithms to address this problem are discussed, which includes the explicit solution for salary_d in terms of the initial condition and d so that a simple calculation is enough to solve the problem.

An algorithm with recursive structure

Algorithm (1-1) uses the recurrence relation where the function is named `DoubledDailySalary` and the `dayWorked` variable is used to represent d, starting from 1 to ndays. As shown in the algorithm below, the `DoubledDailySalary` function calls itself and the base case is the initial condition. This is an example of the recursive algorithm which "is an algorithm that breaks the problem into smaller subproblems and applies the algorithm itself to solve the smaller subproblems" (Lysecky and Vahid 2019: p. 2.7 Recursive definitions).

```
Algorithm (1-1)
Function DoubledDailySalary(dayWorked):
     if dayWorked == 1:
         return 0.01
     else:
         return 2.0 * DoubledDailySalary(dayWorked-1)
End Function
Function Main():
    ndays <- 30
   for i == 1 to ndays DO:
       print('daily salary in day', i, 'is', DoubledDailySalary(i})
11. End Function
```

Algorithm (1-2) is another algorithm which solves the problem. In the algorithm, The `DoubledDailySalary` function which has two arguments named dayWorked and dailySalary is used inside the function.

```
Algorithm (1-2)
 1.  Function DoubledDailySalary(dayWorked, dailySalary):
 2.     if dayWorked < 1 or dayWorked > ndays:
 3.        return
 4.     else:
 5.        Print daily salary on the dayWorked's day
 6.        dayWorked <- dayWorked + 1
 7.        dailySalary <- dailySalary * 2.0
 8.        DoubledDailySalary(dayWorked, dailySalary)
 9.  End Function
 10. Function Main():
 11.    dailySalary <- 0.01
 12.    ndays <- 30
 13.    dayworked <- 1
 14.    DoubledDailySalary(dayWorked, dailySalary)
 15. End Function
```

An algorithm with loop structure

Algorithm (2-1) uses a loop to find the solution. The salary on day one is set to 0.01 and by looping the salary on day two is two times day-one salary. In this way, this problem has the explicit solution of $\text{salary}_d = \text{salary}_0 * (2^d)$. Algorithm (2-2) directly uses the solution so that a simple calculation solve the problem, reducing time.

```
Algorithm (2-1)
```

```
Function Main():
    dailySalary <- 0.01
    ndays <- 30
    for dayWorked == 1 to ndays DO:
        print dailySalary on the dayWorked day
        dailySalary <- dailySalary * 2.0
End Function
```

Algorithm (2-2)

```
Function DoubledDailySalary(dayWorked):
   return 0.01 * (2 ** dayWorked)
End Function
Function Main():
    ndays <- 30
    for i == 1 to ndays DO:
        print('daily salary in day', i, 'is', DoubledDailySalary(i))
```

Computer representation of numbers

The above algorithms requires numbers like *dayWorked* and *dailySalary* as input. The input size in algorithms is discussed as follows: “For many other problems, such as multiplying two integers, the best measure of input size is the total number of bits needed to represent the input in ordinary binary notation” (Cormen et al. 2009: p. 25). The *dayWorked* is a positive integer, which is usually stored with 8, 16, and 32 bits of storage. The range of positive integers with 8 bits of storage is from 0 to 255. The 8-bit singed integer stores integer from -128 to 127. The number that a 32-bit integer can save is 2 to the power of 32 (4,294,967,296). Some languages like Fortran 90/95 can declare the number of bytes to use an integer.

The *real number* like the *dailySalary* which ranges from 0.01 (dollars) to 5,368,709.12 (dollars) in the above problem is usually stored with floating-point format which is based on `scientific notation` to efficiently express small and large numbers. A real number can then be represented as $(-1)^s 2^{c-1023}(1+f)$ or $(-1)^s * 1.f * 2^{c-1023}$ for 64-bit. The 64-bit consists of the following three parts: "The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent, c, called the characteristic, and a 52-bit binary fraction, f, called the mantissa. The base for the exponent is 2"(Burden et al. 2016: p. 15).

A single precision 32-bit float is represented by 1 bit for sign, 8 bits for exponent, 23 bits of fraction. The procedure to convert decimal to binary is as follows: (1) Convert a real number to binary, (2) normalize the binary format, (3) add bias to the exponent part, which is 127 for 32-bit and 1023 for 64-bit by IEEE (Institute for Electrical and Electronic Engineers) 754 *Binary Floating Point Arithmetic Standard*.

Conclusion

Recursion and iteration are commonly used in designing an algorithm. This assignment designs recursive and iterative algorithms each of that shows step-by-step procedure to calculate the doubled daily salary of a worker during the time period of 30 days, starting from the salary of 0.01 (dollars) on day 1. The computational problem, written as a recurrence relation, can be solved using recursion and iteration. An algorithm with the closed-form solution is also provided. It also discusses computer representation of numbers. Implementing the algorithms require number storage of integer and float. The integer number and real number in the problem can be stored with 8-bit (1-byte) and 32-bit single precision float, respectively.

References

Burden, A. M., Burden, R. L., and Faires, J. D. (2016). *Numerical analysis, 10th ed.* Cengage Learning.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms.* The MIT Press.

Lysecky, R. and Vahid, F. (2019). Design and analysis of algorithms. In Lysecky, R. and Vahid, F., editors, *Data structures essential: Pseudocode with python examples.* Zybooks.