**Recurrence Relation and Backward Substitution**

Dr. Byun

09/18/2020

In this note, you will learn the recurrence relation to represent the time efficiency of recursive algorithms and apply the backward substitution technique to solve the recurrence relation.

As a simple example, let's use the recursive function to calculate the factorial values we covered at the last lecture. This is the pseudocode for the factorial function and we will identify the time efficiency of the algorithm.

|  |
| --- |
| 1. Algorithm *F*(n) 2. if n = 0 3. return 1 4. else 5. return F(n - 1) \* n |

Since there is no explicit loop in the pseudocode, it is not clear to identify the time efficiency of the algorithm. The algorithm calls the factorial function F() recursively in the line number 5. To get the time efficiency of the algorithm, we first need to find the basic operation of the pseudocode. In the example, we consider the multiplication operation in the line number 5 as the basic operation. The comparison operation in the line number 2 can also be the basic operation. In this note (and in our textbook), we use the multiplication as the basic operation. Since we identified the basic operation, time efficiency of the algorithm is nothing but to know the number of multiplications executed on the input number n.

To calculate the number of multiplications, we use **a notation** named “**M(n)**” which indicates the “number of multiplications for the input number n”. In the notation, ‘M’ means “**M**ultiplication” and ‘n’ represents the input number **n**.

In other words,

**M(n)**: Number of **multiplications** executed for the input number **n**.

From the meaning of the notation “M(n)”, what is the meaning of “**M(n-1)**”? It means

**M(n-1)**: Number of **multiplications** executed for the input number **n-1**.

Note the difference between the input “n” and “n-1” in the notations.

Similarly, you know that “M(n-2)” represents “Number of **multiplications** executed for the input number **n-2**”.

This way, you know that the notation “M(0)” means the number of multiplications for the input number 0.

**M(0)**: Number of **multiplications** executed for the input number **0**.

Now go back to the original problem of checking the time efficiency of the recursive algorithm. Since the multiplication operation is the basic operation, we should know the actual number of multiplications executed for the input number n which is represented in M(n). Therefore, if we can calculate the value of M(n), we know the category of time efficiency of the algorithm.

From the pseudocode, we know that **M(0) is 0**. That is, if the input value is 0, no multiplication operation is performed. We can express it as follows, and we call it **initial condition**.

M(0) = 0

Now, what is the value of M(n)? We can’t solve it directly because there is a recursive call in the pseudocode at the line number 5. However, we can express it as follows and call it **recurrence relation** (or simply **recurrence**).

M(n) = M(n-1) + 1

It’s very important to understand the meaning of this recurrence relation. It represents that “the number of multiplications executed for the input number n” is the same as “the number of multiplications executed for the input number (n-1) ” plus “one”.

To understand the meaning of the recurrence relation, see the line number 5 again. In the line, there is one multiplication between F(n-1) and n. So, we know that one multiplication is performed here. And then, all remaining multiplications will happen at the function call F(n-1). We can represent it (= number of multiplications for the input number n-1) as M(n-1).

In summary, the following recurrence relation and initial condition must be solved to compute the number of multiplications executed in the pseudo-code.

M(n) = M(n-1) + 1 // recurrence relation

M(0) = 0 // initial condition

To solve it, we use a technique called **backward substitution**.

This is the basic idea of backward substitution.

Since we know the recurrence relation of “M(n) = M(n-1) + 1” for the input number n and n-1, we can represent the relationship between “M(n-1)” and “M(n-2)” as below

**M(n-1) = M(n-2) + 1**

It represents “the number of multiplications executed for the input number n-1” is “the number of multiplications executed for the input number (n-2) ” plus “one”.

Similarly, M(n-2) can be expressed as

**M(n-2) = M(n-3) + 1**

Based on these expressions, we are able to calculate M(n). In the beginning, we start with the recurrence relation as below.

M(n) = M(n-1) + 1

If we replace “M(n-1)” with “M(n-2)+1”, we can represent it as below. Note that we use “[“ and “]” to indicate the replacement.

M(n) = M(n-1) + 1

= [M(n-2) + 1] + 1 // M(n-1) is replaced with “M(n-2) + 1”

= M(n-2) + 2

Now, let’s try to replace “M(n-2)” with “M(n-3) + 1”.

M(n) = M(n-2) + 2

= [M(n-3) + 1] + 2 // M(n-2) is replaced with “M(n-3) + 1”

= M(n-3) + 3

Similarly, we can replace “M(n-3)” with “M(n-4) + 1” like below

M(n) = M(n-3) + 3

= [M(n-4) + 1] + 3

= M(n-4) + 4

If we continue to replace, we can stop at M(0) because we know the initial condition M(0) = 0.

On the other hand, when we perform the replacement, we notice that there are always two terms on the right, such as "M (n-4)" and "4". Based on the result of the substitution, when the argument of the first term is 0, it can be inferred that the second term will be n as below

= M(0) + n

Because M(0) is 0, we can replace M(0) with 0 like below

= 0 + n

= n

Thus, we know that M(n) is n and M(n)(n)

Here is a summary of the backward substitution.

M(n) = M(n-1) + 1

= [M(n-2) + 1] + 1

= M(n-2) + 2

= [M(n-3) + 1] + 2

= M(n-3) + 3

…

= M(n-i) +i // i indicates a general number.

...

= M(0) + n

= 0 + n

= n (n)

**<<< Course Instruction >>>**

**Recurrence relation and backward substitution is one of the most difficult concepts in the whole algorithm class.** Therefore, you should clearly understand it using this simple example.

Now, read **the pages 70-73 (just before Example 2) in the textbook**. After that, continue to read this document.

**Example**

Let’s consider a recursive algorithm for computing the sum of the first n cubes: S(n) = 13 + 23 + … + n3. Set up a recurrence relation and an initial condition for the number of times the algorithm’s basic operation is executed. After that, solve the recurrence relation to calculate the time efficiency of the algorithm.

|  |
| --- |
| 1. Algorithm *S*(n) 2. if n = 1 3. return 1 4. else 5. return S(n - 1) + n \* n \* n |

**Solution**

From the algorithm, we know that the multiplication operation in the line number 5 is the basic operation. Let’s use M(n) to indicate the number of multiplications executed for the input number n. Note that there are two multiplications in the line number 5. We can represent it as

M(n) = M(n-1) + 2 // recurrence relation

As an initial condition, we know that if the input number n is 1, there is no multiplication. We can express it as follows.

M(1) = 0 // initial condition

We can solve the recurrence relation using the backward substitution as below

M(n) = M(n-1) + 2 // replace “M(n-1)” with “M(n-2) +2”

= [M(n-2) + 2] + 2

= M(n-2) + 2 + 2 // replace “M(n-2)” with “M(n-3) +2”

= [M(n-3) + 2] + 2 + 2

= M(n-3) + 2 + 2 + 2

…

= M(n-i) +2\*i // For a general number i, we will get this.

...

= M(1) + 2\*(n-1) // Note that the initial condition M(1) = 0.

= 0 + 2\*(n-1)

= 2\*(n -1) (n)

**Exercise**

Solve the following recurrence relation with the initial condition

X(n) = X(n-1) + 5

X(1) = 1

**Important Note**

Do not see the answer at the next page immediately. Try to solve it by yourself.

**Solution**

We can solve the recurrence relation using the backward substitution as below

X(n) = X(n-1) + 5 // replace “X(n-1)” with “X(n-2) +5”

= [X(n-2) + 5] + 5

= X(n-2) + 5 + 5 // replace “X(n-2)” with “X(n-3) +5”

= [X(n-3) + 5] + 5 + 5

= X(n-3) + 5 + 5 + 5

…

= X(n-i) +5\*i // This is for a general number i.

...

= X(1) + 5\*(n-1) // Initial condition is “X(1) = 1”.

= 1 + 5\*(n-1)

= 5\*(n -1) + 1 (n)