HW #5 피보나치 수열의 계산

자료구조 01분반

전자전기공학부 20196891 이민영

피보나치 수열 구현 c 코드

```c
#include <stdio.h>
#include <time.h>

#define MAX_n 100
#define MAX_FIB 40
int count_rec = 0;
int count_memo = 0;

int fib_rec(int n)        // recusive call 피보나치 수열
{
        count_rec++;     //(3)을 위한 문장
        if (n == 0) return 0;
        if (n == 1) return 1;

        return (fib_rec(n - 1) + fib_rec(n - 2));
}

int fib_memo(int n)       // memoization을 사용한 피보나치 수열을 계산하는 함수
{
        count_memo++;    //(3)을 위한 문장
        static long long memo[MAX_n];
        if (memo[n] > 0)
                return memo[n];
        else if (n == 0)
                return memo[n] = 0;
        else if (n == 1)
                return memo[n] = 1;
        else {
                int f = fib_memo(n - 2) + fib_memo(n - 1);
                memo[n] = f;
                return f;
        }
}

int main() {
        int n;
        int i = 0;

        printf("n = ");
        scanf("%d", &n); //항의 수 입력 받기
        printf("₩n");

        printf("(1) Fibonachi Algorith : recursive call₩n");
        printf("피보나치 수열의 %d 번째 항 : %d₩n ₩n", n, fib_rec(n));
```

```
        printf("(2) Fibonachi Algorith : recursive call with memoization₩n");
        printf("피보나치 수열의 %d 번째 항 : %d₩n ₩n", n, fib_memo(n));

        printf("(3) 두가지 프로그램의 시간 측정₩n");
        clock_t st, et;  //시작 시간과 마침 시간
        for (i = 1; i < MAX_FIB; i++)     //1에서 39까지 1씩 증가시키며 관찰
        {
                printf("=== test case [%d]th₩n", i);
                st = clock();
                fib_rec(i);
                et = clock();
                printf("time_rec : %d₩n", et - st);        //fib_rec 수행 시간
                st = clock();
                fib_memo(i);
                et = clock();
                printf("time_memo : %d₩n", et - st);      //fib_memo 수행 시간
                printf("count_rec: %d  count_memo: %d₩n ₩n", count_rec, count_memo);
        }

        return 0;
}
```

프로그램 구동을 위한 코드에서 각 함수의 역할은 모두 소스 내에 주석으로 적어 두었습니다.

Fib_rec 함수는 결과를 도출하기 위해 수열의 이전 항들의 값을 이용하므로, 함수 내에 같은 함수의 n-1과 n-2 값을 포함하도록 했다. 이 때, fib_rec(0) = 0 , fib_rec(1) = 1 임이 주어졌다.

Fib_memo 함수에서는 한 단계 더 나아가 fib_rec에서 출력되는 값을 순차적으로 배열에 입력해 메모하도록 한다. 이러한 방법을 사용하면 높은 차수의 항을 계산하기 위해 낮은 차수의 항을 여러 번 계산하는 비효율적인 과정을 거칠 필요 없이 이전의 출력값을 memo[]에서 불러오면 되기 때문에 연산의 횟수와 구동 시간이 확연하게 줄어들게 된다. 두 함수의 프로그램 수행 시간과 횟수를 비교한 (3)의 출력 결과를 통해 이러한 효과를 명확하게 확인할 수 있다.

프로그램을 작성하며 memorization 이용 함수에서 처음에 return memo[n] = fib_memo(n-2) + fib_memo(n-1) 를 작성하려 했으나 연산자 오버플로 경고가 발생했다. 이를 해결하기 위해 int형 변수 f 를 생성하고, f에 memo[n]의 값을 입력한 후, f를 return하도록 수정했다.

Recursion을 이용하면서 작성된 함수가 매우 간단해지는 것을 확인할 수 있었고, memoization를 이용해 이를 더욱 효율적으로 이용하는 방법을 터득할 수 있었다. 앞으로 프로그램을 작성하면서 더욱 효육적으로 프로그램을 수행할 수 있는 방법에 대해 좀 더 생각해보면 좋을 것이라는 생각이 들었다.

결과 출력

n = 7

(1) Fibonachi Algorith : recursive call
피보나치 수열의 7 번째 항 : 13

(2) Fibonachi Algorith : recursive call with memoization
피보나치 수열의 7 번째 항 : 13

(3) 두가지 프로그램의 시간 측정
=== test case [1]th
time_rec : 0
time_memo : 0
count_rec: 42 count_memo: 14

=== test case [2]th
time_rec : 0
time_memo : 0
count_rec: 45 count_memo: 15

=== test case [3]th
time_rec : 0
time_memo : 0
count_rec: 50 count_memo: 16

=== test case [4]th
time_rec : 0
time_memo : 0
count_rec: 59 count_memo: 17

=== test case [5]th
time_rec : 0
time_memo : 0
count_rec: 74 count_memo: 18

=== test case [6]th
time_rec : 0
time_memo : 0
count_rec: 99 count_memo: 19

=== test case [7]th
time_rec : 0
time_memo : 0
count_rec: 140 count_memo: 20

=== test case [8]th
time_rec : 0
time_memo : 0
count_rec: 207 count_memo: 23

=== test case [9]th
time_rec : 0
time_memo : 0
count_rec: 316 count_memo: 26

=== test case [10]th
time_rec : 0
time_memo : 0
count_rec: 493 count_memo: 29

=== test case [11]th
time_rec : 0
time_memo : 0
count_rec: 780 count_memo: 32

=== test case [12]th
time_rec : 0
time_memo : 0
count_rec: 1245 count_memo: 35

=== test case [13]th
time_rec : 0
time_memo : 0
count_rec: 1998 count_memo: 38

=== test case [14]th
time_rec : 0
time_memo : 0
count_rec: 3217 count_memo: 41

=== test case [15]th
time_rec : 0
time_memo : 0
count_rec: 5190 count_memo: 44

=== test case [16]th
time_rec : 0
time_memo : 0
count_rec: 8383 count_memo: 47

=== test case [17]th
time_rec : 0
time_memo : 0
count_rec: 13550 count_memo: 50

=== test case [18]th
time_rec : 0
time_memo : 0
count_rec: 21911 count_memo: 53

=== test case [19]th
time_rec : 1
time_memo : 0
count_rec: 35440 count_memo: 56

=== test case [20]th
time_rec : 1
time_memo : 0
count_rec: 57331 count_memo: 59

=== test case [21]th
time_rec : 1
time_memo : 0
count_rec: 92752 count_memo: 62

=== test case [22]th
time_rec : 3
time_memo : 0
count_rec: 150065 count_memo: 65

=== test case [23]th
time_rec : 4
time_memo : 0
count_rec: 242800 count_memo: 68

=== test case [24]th
time_rec : 6
time_memo : 0
count_rec: 392849 count_memo: 71

=== test case [25]th
time_rec : 10
time_memo : 0
count_rec: 635634 count_memo: 74

=== test case [26]th
time_rec : 16
time_memo : 0
count_rec: 1028469 count_memo: 77

=== test case [27]th
time_rec : 28
time_memo : 0
count_rec: 1664090 count_memo: 80

=== test case [28]th
time_rec : 42
time_memo : 0
count_rec: 2692547 count_memo: 83

=== test case [29]th
time_rec : 70

time_memo : 0
count_rec: 4356626 count_memo: 86

=== test case [30]th
time_rec : 112
time_memo : 0
count_rec: 7049163 count_memo: 89

=== test case [31]th
time_rec : 168
time_memo : 0
count_rec: 11405780 count_memo: 92

=== test case [32]th
time_rec : 331
time_memo : 0
count_rec: 18454935 count_memo: 95

=== test case [33]th
time_rec : 443
time_memo : 0
count_rec: 29860708 count_memo: 98

=== test case [34]th
time_rec : 763
time_memo : 0
count_rec: 48315637 count_memo: 101

=== test case [35]th
time_rec : 1156
time_memo : 0
count_rec: 78176340 count_memo: 104

=== test case [36]th
time_rec : 1968
time_memo : 0
count_rec: 126491973 count_memo: 107

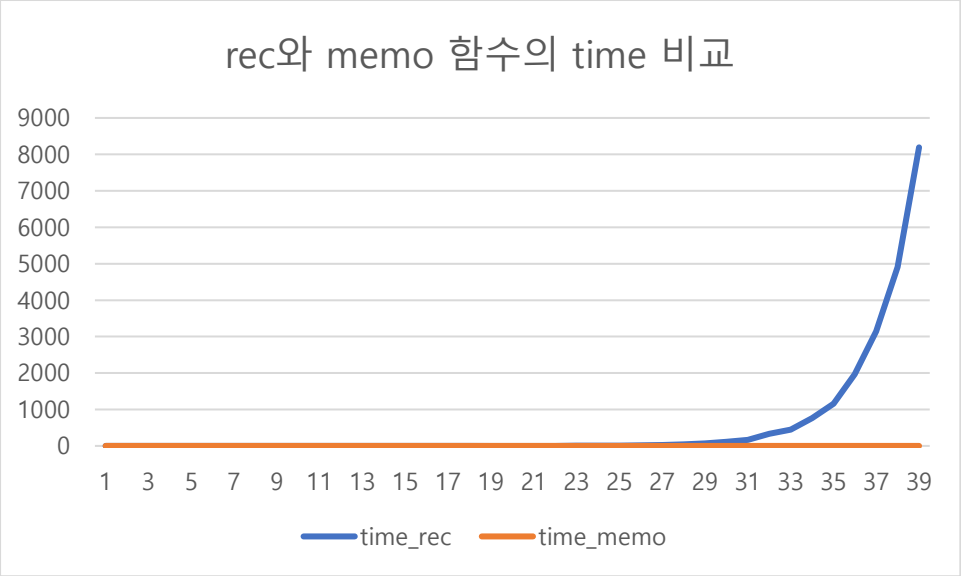
=== test case [37]th
time_rec : 3144
time_memo : 0
count_rec: 204668310 count_memo: 110

=== test case [38]th
time_rec : 4911
time_memo : 0
count_rec: 331160281 count_memo: 113

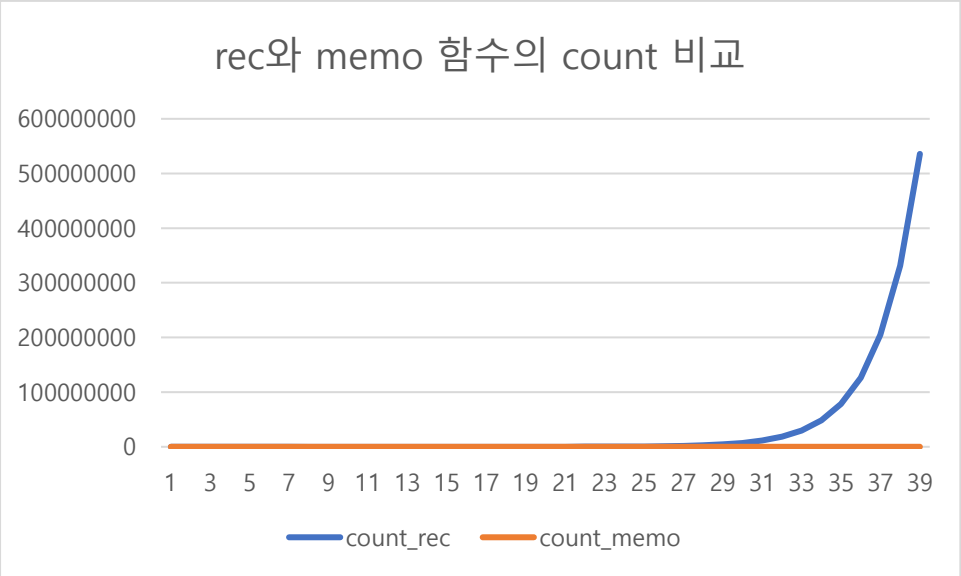
=== test case [39]th
time_rec : 8193
time_memo : 0
count_rec: 535828590 count_memo: 116

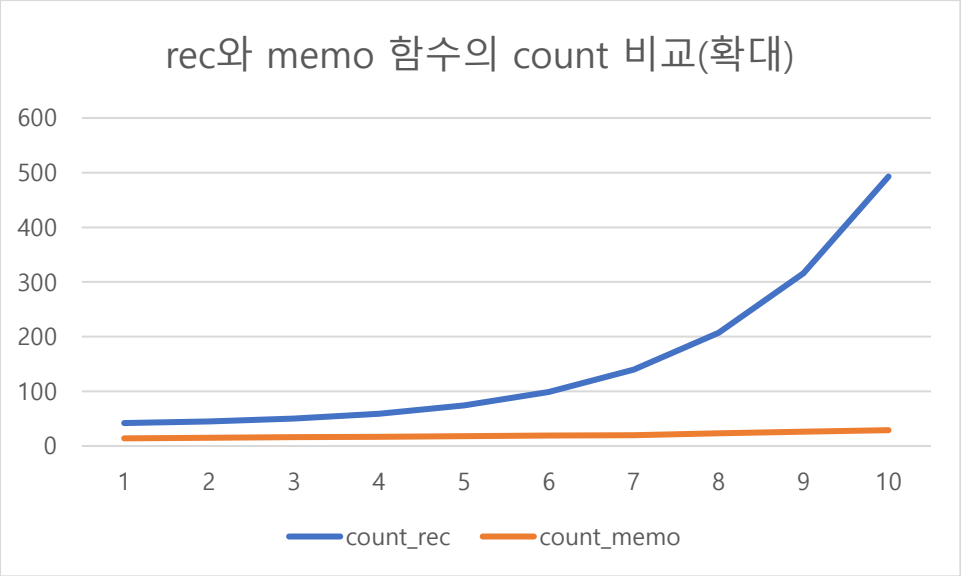
이 창을 닫으려면 아무 키나 누르세요...

(4) 수행의 값들로 그린 그래프 관찰의견



수행시간을 살펴보았을 때, memoization을 이용한 fib_memo 함수는 수행의 끝까지 0으로 나타난 것으로 보아 실행 시간이 매우 짧음을 알 수 있다. 반면 recursive call만을 이용한 fib_rec 함수의 경우, 약 27번째 실행까지는 time_rec과 큰 차이가 없지만, 이후로는 지수함수의 형태를 띄며 급격히 수행시간이 증가하는 것을 확인했다. 이로써 memoize를 이용하는 것이 훨씬 효율적으로 재귀함수를 이용할 수 있다는 사실을 확인할 수 있었다.





추가적으로 n번째 항을 구하기 위해 수행되는 연산의 횟수를 살펴보면 count_rec와 count_memo 모두 n이 증가할수록 증가하지만, recursive call 만 이용했을 때에는 횟수가 지수함수의 형태로 매우 큰 수까지 증가하는 반면, memoize를 이용했을 때에는 상대적으로 매우 적은 수행 횟수만으로 결과를 출력할 수 있음을 확인했다.