2018920065 루안리치 자료구조 과제 #1

#순환을 이용한 방법

```
1  //
2  //  main.cpp
3  //  ds_w3_hanoi
4  //
5  //  2018920065 luan li chi
6
7
8  #include <iostream>
9  #include <stdio.h>
10 #include <time.h>
11
12 //2018920065 루안리치 자료구조 과제1 : 하노이탑 순환적인 방법
13 void hanoi_tower(int n, char from, char tmp, char to){
14     if(n==1) printf("원판 1을 %c 에서 %c 으로 옮긴다.\n",from,to);
15     else{
16         hanoi_tower(n-1, from, to, tmp);
17         printf("원판 %d을 %c 에서 %c 으로 옮긴다.\n",n,from,to);
18         hanoi_tower(n-1, tmp, from, to);
19     }
20 }
21
22 int main(void){
23     clock_t start, stop;
24     double duration;
25     start=clock();
26     hanoi_tower(4,'A','B','C');
27     stop=clock();
28     duration=(double)(stop-start)/CLOCKS_PER_SEC;
29     printf("time: %f sec\n",duration);
30     return 0;
31 }
32 //2018920065 루안리치 자료구조 과제1 : 하노이탑 순환적인 방법
33
```

순환법 사용한 결과:

```
32 //2018920065 루안리치 자료구조 과제1 : 하노이탑 순환적인 방법
33
```

```
원판 1을 A 에서 B 으로 옮긴다.
원판 2을 A 에서 C 으로 옮긴다.
원판 1을 B 에서 C 으로 옮긴다.
원판 3을 A 에서 B 으로 옮긴다.
원판 1을 C 에서 A 으로 옮긴다.
원판 2을 C 에서 B 으로 옮긴다.
원판 1을 A 에서 B 으로 옮긴다.
원판 4을 A 에서 C 으로 옮긴다.
원판 1을 B 에서 C 으로 옮긴다.
원판 2을 B 에서 A 으로 옮긴다.
원판 1을 C 에서 A 으로 옮긴다.
원판 3을 B 에서 C 으로 옮긴다.
원판 1을 A 에서 B 으로 옮긴다.
원판 2을 A 에서 C 으로 옮긴다.
원판 1을 B 에서 C 으로 옮긴다.
time: 0.000031 sec
Program ended with exit code: 0
```

#반복을 이용한 방법

```cpp
//
//  main.cpp
//  ds_w3_hanoi_iter
//
//  2018920065 luan li chi


#include <iostream>
#include <stdio.h>
#include <time.h>
#define pause getch()
#define is_stack_empty()  (top < 0)
#define MAX 100

//2018920065 루안리치 자료구조 과제1 : 하노이탑 반복적인 방법
int top;
int stack[MAX];

void init_stack()
{
    top = -1;
}

int push(int val)
{
    if(top >= MAX -1)  // Overflow
    {
       printf("\nOverflow.");
       return -1;
    }
    stack[++top] = val;
    return val;
}

int pop()
{
    if(top < 0)  // Underflow
    {
       printf("\nUnderflow.");
       return -1;
    }
    return stack[top--];
}

void move(int n, char a, char c)
{
    printf("원판 %d을 %c 에서 %c 으로 옮긴다\n", n, a, c);
}

void hanoi_iter(int n, char from, char tmp, char to)
{
   int done = 0;
   init_stack();   // 초기화
   while(!done)
   {
      while(n > 1)
      {
          push(to);
          push(tmp);
          push(from);
          push(n);
          n--;
          push(to);
          to = tmp;
          tmp = pop();
      }

      move(n, from, to);  // 종료
```

```
69        if(!is_stack_empty()) // (top < 0) -> !(top < 0)
70        {
71            n = pop();
72            from = pop();
73            tmp = pop();
74            to = pop();
75            move(n, from, to);
76            n--;
77            push(from);
78            from = tmp;
79            tmp = pop();
80         }
81         else
82            done = 1;  // 스택이 비면 끝
83     }
84 }
85
86
87
88 int main(void){
89     clock_t start, stop;
90     double duration;
91     start=clock();
92     hanoi_iter(4,'A','B','C');
93     stop=clock();
94     duration=(double)(stop-start)/CLOCKS_PER_SEC;
95     printf("time: %f sec\n",duration);
96     return 0;
97 }
98 //2018920065 루안리치 자료구조 과제1 : 하노이탑 반복적인 방법
99
```

반복법 사용한 결과:

```
98  //2018920065 루안리치 자료구조 과제1 : 하노이탑 반복적인 방법
]
원판 1을 A 에서 B 으로 옮긴다
원판 2을 A 에서 C 으로 옮긴다
원판 1을 B 에서 C 으로 옮긴다
원판 3을 A 에서 B 으로 옮긴다
원판 1을 C 에서 A 으로 옮긴다
원판 2을 C 에서 B 으로 옮긴다
원판 1을 A 에서 B 으로 옮긴다
원판 4을 A 에서 C 으로 옮긴다
원판 1을 B 에서 C 으로 옮긴다
원판 2을 B 에서 A 으로 옮긴다
원판 1을 C 에서 A 으로 옮긴다
원판 3을 B 에서 C 으로 옮긴다
원판 1을 A 에서 B 으로 옮긴다
원판 2을 A 에서 C 으로 옮긴다
원판 1을 B 에서 C 으로 옮긴다
time: 0.000045 sec
Program ended with exit code: 0
```

#시간복잡도

원판 4 개 일때

〈12131214,1213,12,1〉

$$2^3 + 2^2 + 2^1 + 2^0 = 2^4 - 1$$

총 $2^4 - 1$ 번 이동으로 원판을 모두 이동 시킬 수 있다.

원판 5 개 일때

〈12131214121312151,12131214,1213,12,1〉

$$2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 2^5 - 1$$

총 $2^5 - 1$ 번 이동으로 원판을 모두 이동 시킬 수 있다.

정리하면

원판 n 개 일때, $T(n) = 2^n - 1$

시간복잡도 = $\boldsymbol{O(2^n)}$

실제실행결과를 보면,

원판수	순환법 시간 (초)	반복법 시간 (초)
4	0.000031	0.000045
10	0.002077	0.002029
20	1.633322	1.671304

둘 방법의 소요시간 차이가 크지 않지만, 코드의 구현은 순환법이 훨씬 더 깔끔하고 이해하기 쉬워서, 순환법이 더 좋다고 생각합니다.