제목 : 하노이 타워 기술보고서
학과 : 컴퓨터학부
학번 : 20172605
이름 : 김도현
과목 : 컴퓨터 수학
담당교수 : 길아라
작성일자 : 2017.5.3

목차

1. 서론

분야 : 하노이 타워 문제는 1883년 프랑스 수학자 Lucas가 고안한 문제로, 세 개의 기둥이 주어지고, 1번 기둥에 놓인 크기가 다른 원반들을 전부 3번 기둥으로 옮기는 문제이다. 이 때, 작은 원반 위에 큰 원반이 놓일 수 없고, 한 번에 한 개의 원반만 옮길 수 있다는 조건이 따른다. 그리고 원반 이동의 최소 횟수를 보장하여야 한다.
동기 : 재귀적 알고리즘에 흥미가 있어서 하노이 타워 문제의 해결방법을 공부하고 싶었다.
목적 : 하노이 타워의 재귀적 알고리즘을 작성하고, 그 알고리즘의 진행과정을 시각화한다.
연구방법 : 인터넷 검색, 프로그래밍, 디버깅

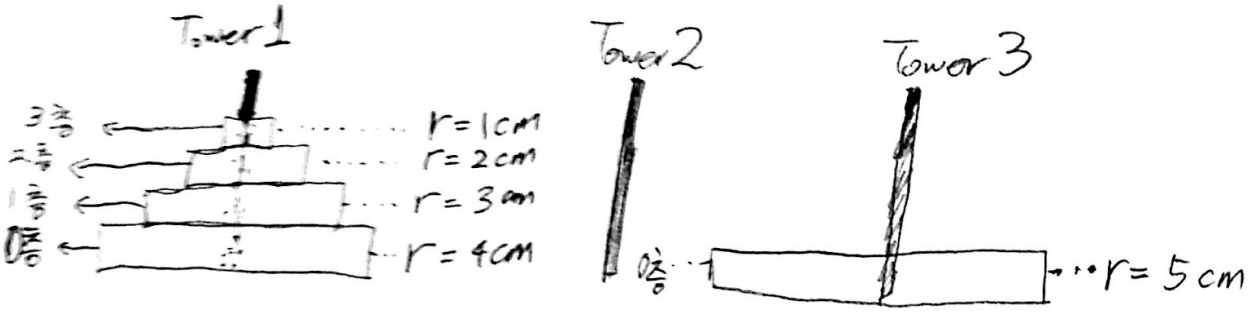
2. 관련 연구

"하노이 탑 프로그래밍 과정에서 나타나는 사고 패턴에 관한 심층 분석"(한국컴퓨터교육학회, 윤지현)
"하노이 탑의 구조 탐색을 위한 초등 수학 영재 학습 자료 개발"(경인교육대학교, 오민아)
이미 해결된 문제이기 때문에 알고리즘 교육에 활용하는 방향의 연구가 많이 이루어지고 있다.

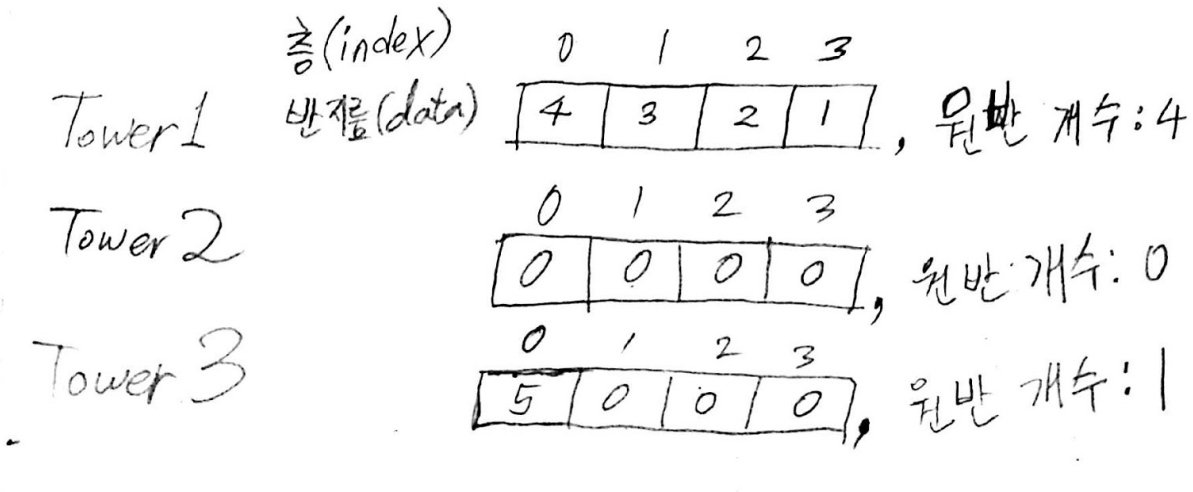
3. 본론-설계

3.a 자료형 설계

3.a.i 기본 데이터



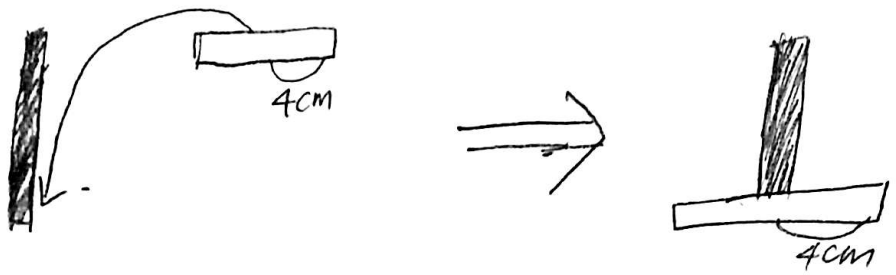
3.a.ii 데이터 추상화



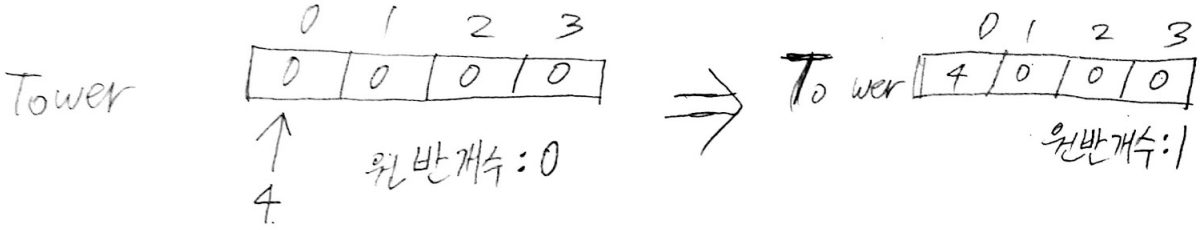
이로부터 Tower={원반을 담을 배열, 담겨져 있는 원반의 개수}라는 데이터를 설계할 수 있다.

3.b Push 기능 설계

3.b.i Push 기본 동작

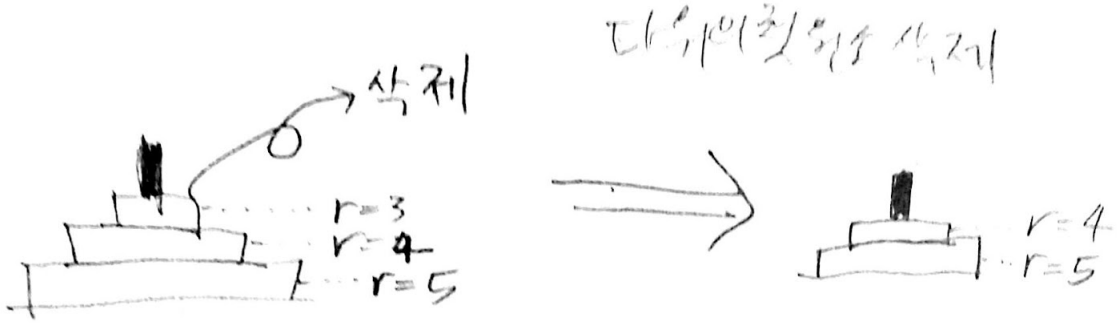


3.b.ii Push 동작 추상화

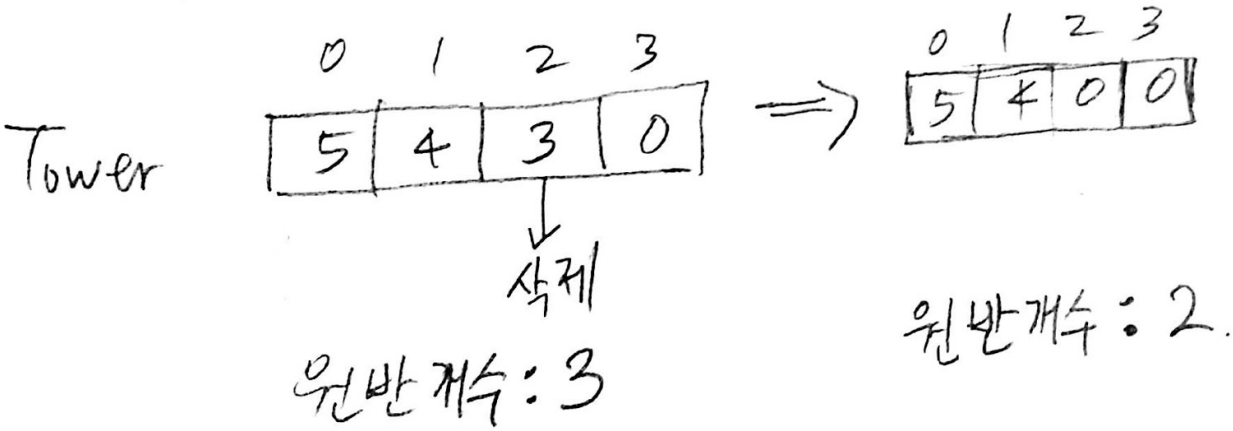


3.c Pop 기능 설계

3.c.i Pop 기본 동작

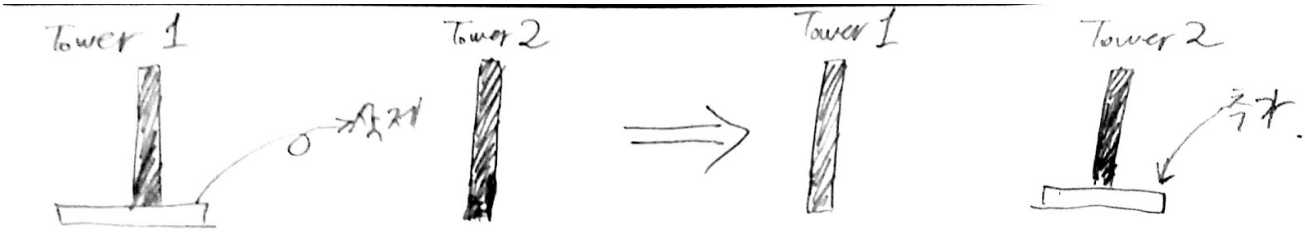


3.c.ii Pop 동작 추상화



3.d Move 기능 설계

3.d.i Move 기본 동작



3.d.ii Move 동작 추상화

from타워에서 pop한 원반을 to타워에 push한다.

3.e Solve 기능 설계

3.e.i Solve 문제 분해

기본문제)

n개의 원반을 탑1에서 탑3으로 이동

분해한 문제)

1. 탑1의 원반 n-1개를 탑2로 이동
2. 원반n을 탑1->탑3으로 이동
3. 탑2의 원반 n-1개를 탑3으로 이동

3.e.ii Solve 원반 이동 횟수

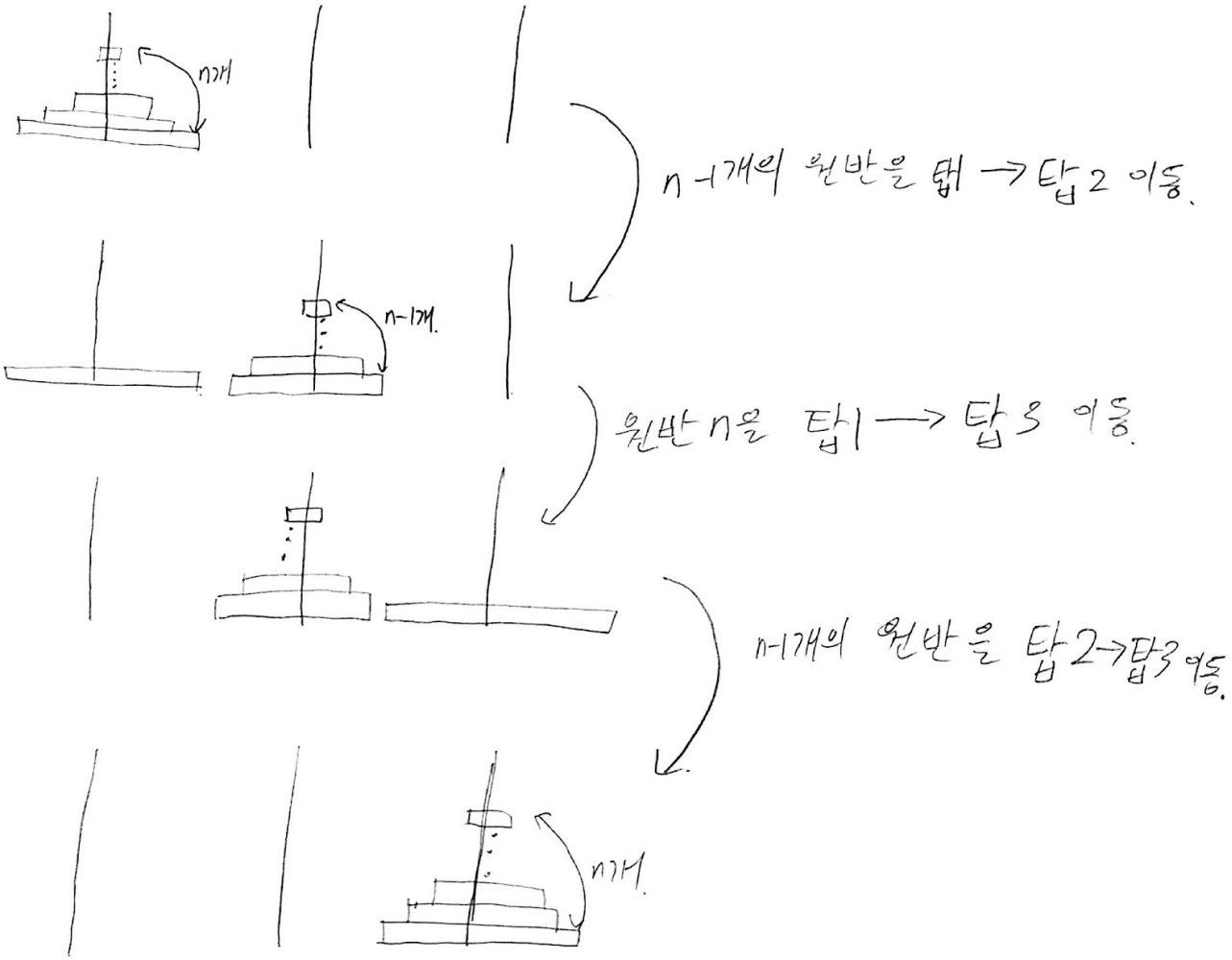
기본단계)

n=0일 때 원반이동이 없으므로 0을 반환한다.

귀납단계)

그 외의 경우 Solve(1, 2, n-1)+1+Solve(2, 3, n-1)을 반환한다.

3.e.iii Solve 동작 시각화



3.f Print 기능 설계

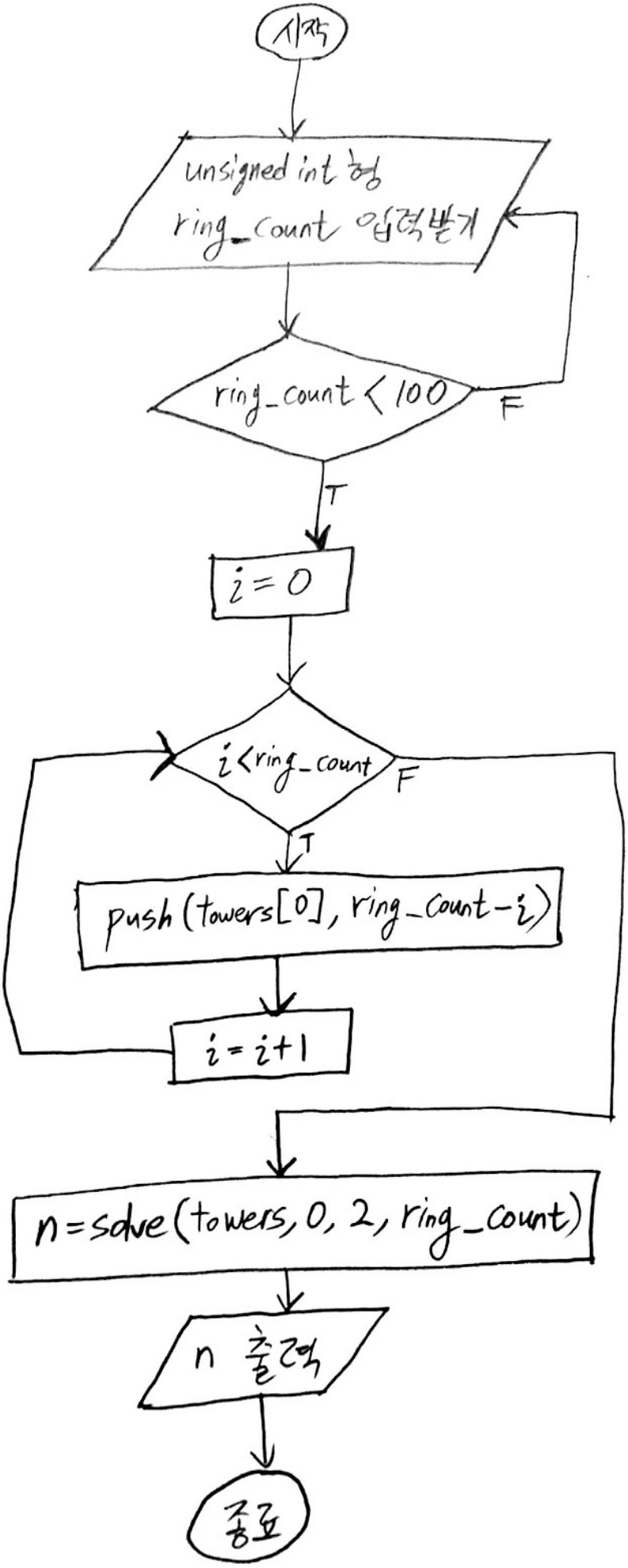
3.f.i Print 기본 동작

원반을 옮기는 과정을 보여줘야 한다.

3.f.ii 동작 구체화

1. 어떤 크기의 원반을 어느 타워에서 어느 타워로 옮기는지 출력한다.
2. 현재 단계에서 각 3개의 타워에 존재하는 원반들을 출력한다.
(원반 대신에 원반의 반지름만 시각화해도 동일하게 인식 가능함)
(∵추상화 데이터가 동일하기 때문)
3. 화면에 바로 출력하지 않고 2차원 char 배열에 저장하고 출력하면 형태구성이 쉽다.
4. 해결과정을 출력하기 위해서는 Move함수에서 Print함수를 호출해야 한다.

3.g main 함수 flowchart



4. 구현 또는 모의실험
사용한 기기 : Hansung H57 DGA7700
시스템 구성 : Windows 10 + Visual Studio 2015 Community
※ Windows Server 2008 R2 Enterprise(N Cloud) + Cygwin 및 Ubuntu + gcc 환경에서도 정상동작함을 확인함.
주의사항 : 원반을 10개 이상 사용하면 처리시간이 오래걸린다.
계산방법 : 컴파일 및 링크과정 후 나온 실행파일을 실행하고, 원반의 개수를 입력하면 결과가 출력된다.

4.a 데이터 추상화 C언어 구현

```c
struct Tower
{
    int* rings;
    int count;
};
```

4.b Push 기능 C언어 구현

```c
void push(struct Tower* tower, int n)
{
    tower->rings[tower->count++] = n;
}
```

4.c Pop 기능 C언어 구현

```c
int pop(struct Tower* tower)
{
    return tower->rings[--tower->count];
}
```

4.d Move 기능 C언어 구현

```c
void move(struct Tower towers[TOWER_COUNT], int from, int to)
{
    int n;
    push(&towers[to], n = pop(&towers[from]));
    printf("원반%d을(를) 탑%d에서 탑%d로 옮긴다.\n", n, from + 1, to + 1);
    print(towers);
}
```

4.e Solve 기능 C언어 구현

```c
int solve(struct Tower towers[TOWER_COUNT], int from, int to, int n)
{
    if (n <= 0)
        return 0;
    int i, tmp;
    for (i = 0; i < TOWER_COUNT; i++)
        if (i != from && i != to)
            tmp = i;
    int n1 = solve(towers, from, tmp, n - 1);
    move(towers, from, to);
    int n2 = solve(towers, tmp, to, n - 1);
    return n1 + n2 + 1;
}
```

4.e.i Solve 함수의 정당성 증명

기본단계)

Solve(towers, from, to, 0)는 옮길 원반이 0개이므로, 그 자체로 모든 원반을 to타워로 옮긴 것이고, 원반이동도 0번 일어났으니 최소다. 따라서 기본단계가 완성된다.

귀납단계)

귀납단계가정 : 타워 a, b에 대하여 a≠b일 때, Solve(towers, a, b, k)는 a타워의 원반 k개를 b타워에 최소횟수로 이동시킨다. 즉, 전칭한정 정의역 = {타워0, 타워1, 타워2}이고 ∀a∀b((a≠b)→Solve(towers, a, b, k))

귀납단계를 완성하기 위해 귀납단계가정이 성립할 때, a타워의 원반 k+1개를 b타워에 최소이동횟수로 옮길 수 있음을 증명하자. 즉, ∀a∀b((a≠b)→Solve(towers, a, b, k))→ ∀a∀b((a≠b)→Solve(towers, a, b, k))가 성립함을 보이면 귀납단계가 완성된다.

시작타워를 from, 도착타워를 to, 남은타워를 tmp라고 하자.

귀납단계가정에 의해 Solve(towers, from, tmp, k)가 성립한다. 따라서 from에 있는 k+1개의 원반 중에 맨 아래의 원반을 제외한 k개의 원반을 최소이동횟수로 tmp에 옮길 수 있다.

이제 from타워에 한 개의 원반만 있으므로 문제에 주어진 조건에 따라 그 한 개의 원반은 to로 한 번에 옮길 수 있으며, 자명하게 최소이동횟수이다.

귀납단계가정에 의해 Solve(towers, tmp, to, k)가 성립한다. 따라서 tmp에 옮겨뒀던 k개의 원반을 최소이동횟수로 to에 옮길 수 있다.

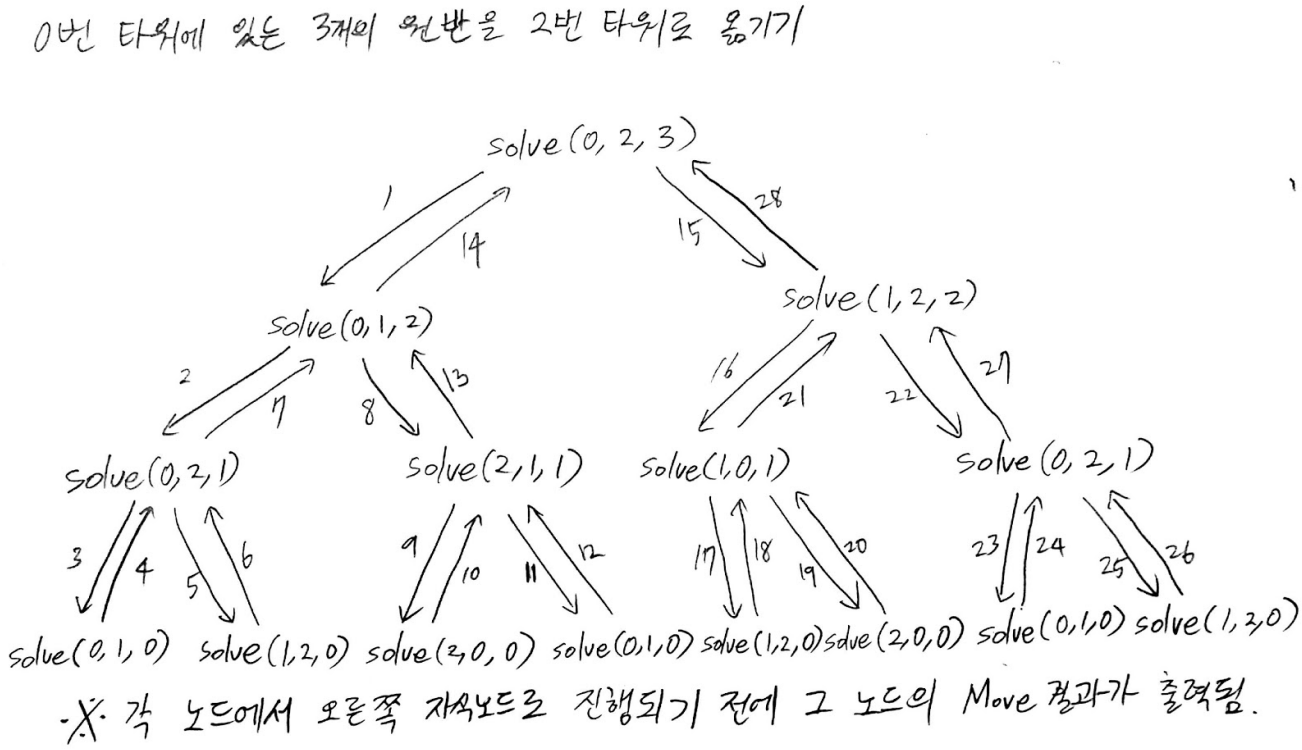
이제 k+1개의 원반이 모두 from에서 to로 옮겨졌으며, 각 단계의 이동횟수가 최소이므로 그 합이 최소이동횟수이다. 따라서 귀납단계가 완성된다.

∴ 기본단계와 귀납단계가 완성됐으므로, 수학적 귀납법에 의해 solve는 모든 자연수에 대해 잘 작동한다.

4.e.ii Solve 함수의 시간복잡도

k=0이 아닐 때 Solve 호출마다 Move가 1번 호출되고, k=0일 때 Solve 호출마다 Move가 0번 호출된다. 원반이 n개일 때, Solve는 총 2^(n+1)-1번 호출되고, 그 중에서 k=0인 Solve는 총 2^n번 호출되므로, Move는 총 {2^(n+1)-1}-{2^n}=2^n-1번 호출된다. 따라서 원반이동을 기준으로 시간복잡도는 O(2^n)이다.

4.e.iii Solve 함수 실행과정 따라가보기



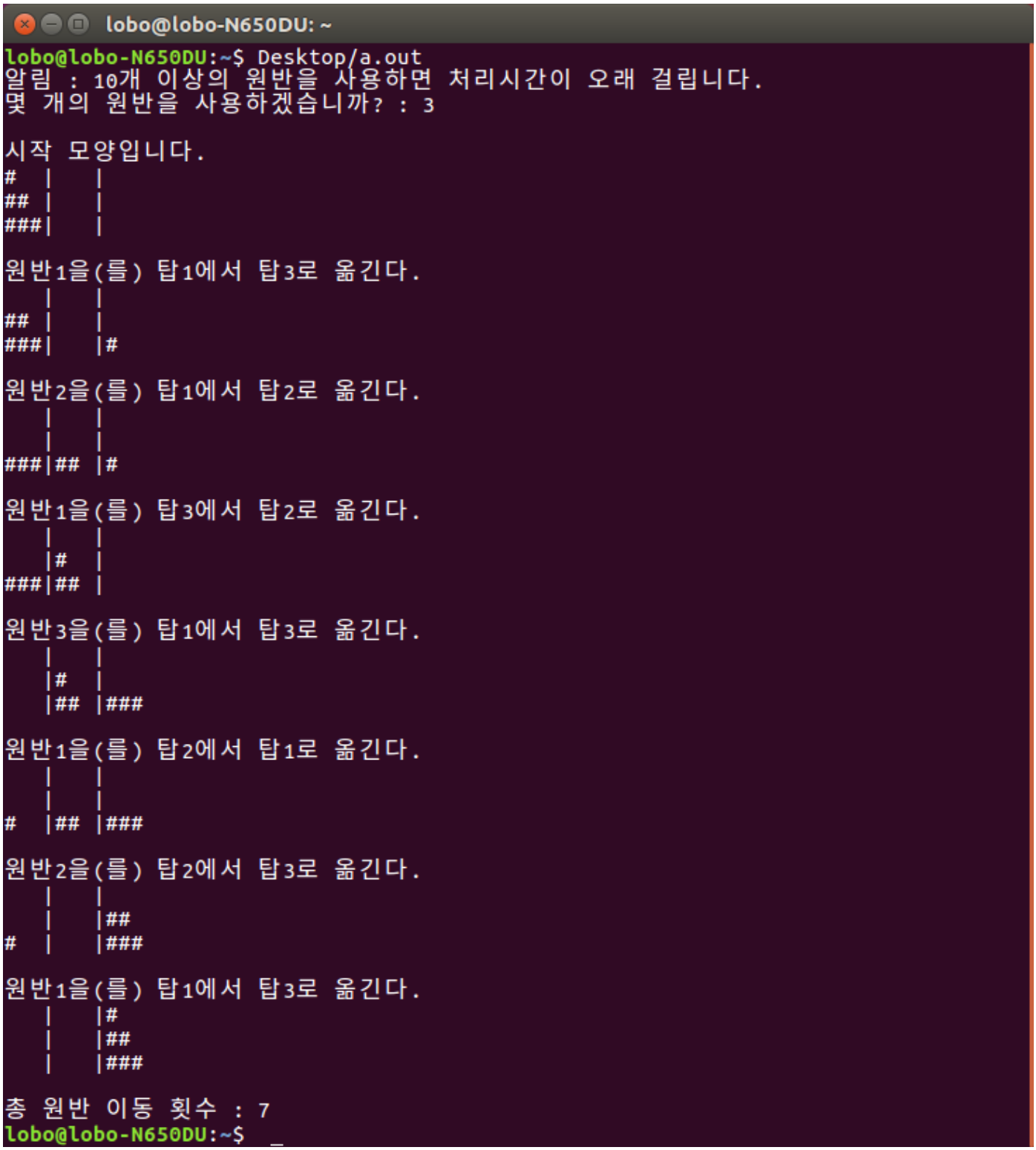
4.f Print 기능 C언어 구현

```c
void print(struct Tower towers[TOWER_COUNT])
{
    unsigned print_space_x = ring_count * 3 + 2 + 1;
    unsigned print_space_y = ring_count;
    static char** space;
    if (!space)
    {
        space = malloc(sizeof(char*)*print_space_y);
        int i;
        for (i = 0; i < print_space_y; i++)
            space[i] = malloc(sizeof(char)*print_space_x);
    }
    int i, j, k;
    for (i = 0; i < print_space_y; i++)
        memset(space[i], ' ', sizeof(char)*print_space_x);
    for (i = 0; i < print_space_y; i++)
    {
        space[i][print_space_x - 1] = '\0';
        space[i][ring_count + (1 * 0)] = '|';
        space[i][ring_count * 2 + (1 * 1)] = '|';
    }
    for (i = 0; i < TOWER_COUNT; i++)
    {
        for (j = 0; j < towers[i].count; j++)
        {
            for (k = 0; k < towers[i].rings[j]; k++)
            {
                space[print_space_y - 1 - j][ring_count*i + i + k] = '#';
            }
        }
    }
    for (i = 0; i < print_space_y; i++)
        printf("%s\n", space[i]);
    printf("\n");
}
```

4.g main 함수 C언어 구현

```c
int main(void)
{
    while (1)
    {
        printf("알림 : 10개 이상의 원반을 사용하면 처리시간이 오래 걸립니다.\n");
        printf("몇 개의 원반을 사용하겠습니까? : ");
        scanf("%u", &ring_count);
        if (ring_count < 100)
            break;
        printf("오류 : 0 ~ 100 사이의 수를 입력해주세요.\n");
    }
    printf("\n");
    struct Tower towers[TOWER_COUNT] = { 0, };
    int i;
    for (i = 0; i < TOWER_COUNT; i++)
    {
        towers[i].rings = malloc(sizeof(int)*ring_count);
        memset(towers[i].rings, 0, sizeof(int)*ring_count);
    }
    for (i = 0; i < ring_count; i++)
        push(towers + 0, ring_count - i);
    printf("시작 모양입니다.\n");
    print(towers);
    int n = solve(towers, 0, 2, ring_count);
    printf("총 원반 이동 횟수 : %d\n", n);
    for (i = 0; i < TOWER_COUNT; i++)
        free(towers[i].rings);
    return 0;
}
```

4.h 프로그램의 출력 결과



5. 결론, 향후 연구방향

프로그램이 하노이 타워 문제의 해결과정을 시각화하여 보여준다. 현재 하노이 타워는 컴퓨터 교육분야에서 연구가 되고 있으므로, 하노이탑 해결과정 시각화가 재귀적 알고리즘 교육에 활용이 된다면 많은 도움이 될 것이다.