**[제한 사항]**

|  |  |
| --- | --- |
| 시간 | 25개 테스트케이스를 합쳐서 C++ 의 경우 3초 / Java 의 경우 3초 |
| 메모리 | 힙, 정적 메모리 합쳐서 256MB 이내, 스택 메모리 1MB 이내 |

**※** 본 문제는 임직원의 S/W 문제해결 역량 향상을 위한 **Professional 실전형 연습문제**이며

함께 제공되는 **Testcase 또한 학습을 위한 연습용**입니다.

|  |
| --- |
| **①   C 또는 C++로 답안을 작성하시는 응시자께서는 검정시스템에 제출 시, Language 에서 C++ 를 선택하신 후 제출하시기 바랍니다.**  **②   Main 과 User Code 부분으로 구성되어 있습니다.**  **ㅇ   A.**Main         : 수정할 수 없는 코드이며, 채점 시 비 정상적인 답안 검출 등 평가를 위한 로직이 추가 될 수 있습니다.  **ㅇ   B.**User Code  : 실제 응시자가 작성해야 하는 코드이며, 제출 시에는 표준 입출력 함수가 포함되어 있으면 안 됩니다.  **③   Local PC 에서 프로그래밍 시 유의 사항**  **ㅇ   A.**2개의 파일을 생성하셔야 합니다. ( main.cpp / solution.cpp 또는 Solution.java / UserSolution.java )  **ㅇ   B.**Main 부분의 코드를 main.cpp 또는 Solution.java 에 복사해서 사용하시기 바랍니다.  **ㅇ   C.**sample\_input.txt 를 사용하시기 위해서는 Main 부분의 코드 내에  **ㅇ   C.**표준 입력을 파일로 전환하는 코드 ( 주석처리 되어 있음 ) 의 주석을 풀어서 사용하시면 됩니다.  **ㅇ   D.**User Code 부분의 코드를 작성하신 후 서버에 제출하실 때,  **ㅇ   D.**디버깅을 위한 표준 입출력 함수를 모두 삭제 또는 주석 처리해 주셔야 합니다.  **④   문제 내에 제약조건을 모두 명시하지 않으므로 주어지는 코드를 분석하셔야 합니다.**  **⑤   코드는 개발 언어에 따라 상이할 수 있으므로, 작성할 언어를 기준으로 분석하셔야 합니다.** |

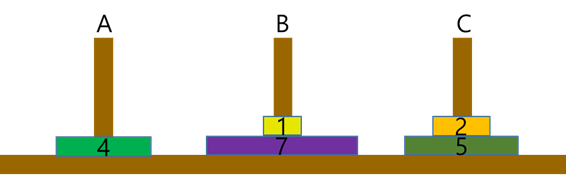
**[문제 설명]**

|  |  |
| --- | --- |
| 하노이탑에 대한 전설에 의하면 인도의 큰 고대 사원에서 64개의 원판이 본래의 자리를 떠나 다른 한 기둥으로 모두 옮겨지면 탑과 사원 승려들이 모두 먼지가 되어 사라지면서 세상의 종말이 온다고 한다.    하노이탑을 옮기는 규칙은 아래와 같다.  - 한번에 한 개의 원판만을 옮긴다.  - 큰 원판을 작은 원판 위에 놓을 수 없다.  - 원판은 세 기둥 중에 어느 한 기둥에 꽂혀야 하며 바닥에 놓아서는 안된다.  기둥 A에 원판이 모두 있는 경우의 하노이 타워의 의사코드는 아래와 같다.   |  | | --- | | Hanoi(from, by, to, n)  \_\_\_\_if n<=0  \_\_\_\_\_\_\_\_return  \_\_\_\_Hanoi(from, to, by, n-1)  \_\_\_\_Move(from, to)  \_\_\_\_Hanoi(by, from, to, n-1) |   n번째 원판이 이동하기 위해서, n-1 번째 원판을 n번째 원판이 있는 위치와 이동해야 하는 위치가 아닌 기둥으로 이동시킨 후, n번째 원판을 이동시키는 원리이다. |

당신은 어느 날 공원을 걷다가 공원에 설치된 하노이탑 놀이시설을 발견한다.

관리 상태가 좋지 않아서, 부서진 원판과 없어진 원판들도 보인다.

모든 원판들이 있지는 않지만, 방금 전까지 놀다간 흔적이 보인다.

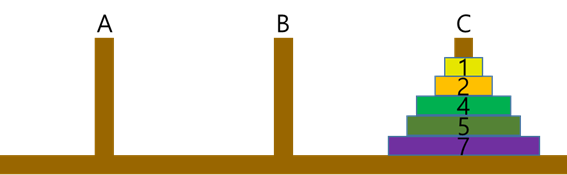


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_[Fig. 1]**

[Fig. 1] 과 같이 크기 3, 크기 6 원판은 보이지 않지만, 잠깐 머리를 식히기에는 충분하다.

현재 상태에서 하노이의 규칙을 지키면서 최소 횟수로 이동하여 모든 원판을 기둥 C로 옮기고 싶다.

최종 목표 상태는 [Fig. 2] 이다.



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [Fig. 2]**

당신은 매우 꼼꼼하고 완벽주의를 지향하는 사람이므로 최소 이동 횟수로 기둥 C로 가는 규칙이 없는지 고민하기 시작했다. 얼마간의 시간이 흐른 뒤 당신은 무릎을 치며 기뻐했다.

어떠한 상태에서도 최단 횟수로 기둥 C로 모든 원판을 옮기기 위한 순서는 유일하다라는 것을 알았다.

**그리고, 크기가 가장 큰 원판부터 가장 작은 원판까지 차례대로 현재의 위치와 그 원판이 이동해야 하는 위치를 파악하면 바로 위의 원판이 어디로 이동해야 하는 지 결정됨을 깨달았다.**

누락된 원판이 있는 하노이탑의 각 기둥의 현재 상태가 주어지면 최소 횟수로 기둥 C로 이동하기 위한 중간 과정을 반환하는 프로그램을 작성하라.

아래 API 설명을 참조하여 각 함수를 구현하라.

※ 아래 함수 signature는 C/C++에 대한 것으로 다른 언어에 대해서는 제공되는 Main과 User Code를 참고하라.

아래는 User Code 부분에 작성해야 하는 API 의 설명이다.

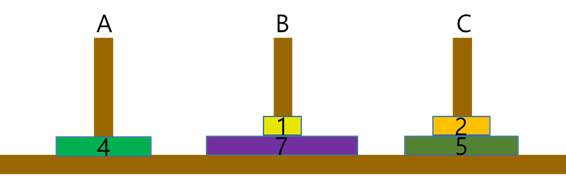
|  |
| --- |
| **void init(int N[3], int mDisk[3][])** |
| 각 테스트 케이스의 처음에 호출된다.  N[0]은 기둥 A 의 원판의 수, N[1]은 기둥 B의 원판의 수, N[2]은 기둥 C의 원판의 수이다.  mDisk[0]에는 기둥 A에 놓여있는 N[0]개의 원판들의 크기가 오름차순으로 주어진다.  mDisk[1]에는 기둥 B에 놓여있는 N[1]개의 원판들의 크기가 오름차순으로 주어진다.  mDisk[2]에는 기둥 C에 놓여있는 N[2]개의 원판들의 크기가 오름차순으로 주어진다.  mDisk[][0]이 각 기둥에서 가장 작은 원판이다.  예를 들어 기둥 A에 크기가 3, 5, 6 인 원판들이 놓여있는 경우 mDisk[0][]는 {3, 5, 6}으로 주어진다.  기둥 C 가 목표 기둥이다.  원판의 총 개수는 1000개 이하이며, 원판의 크기는 모두 다르다.    ***\_\_Parameters***  \_\_\_\_\_N : 각 기둥의 원판 개수 ( 0 ≤ N[] ≤ 1000, 5 ≤ N[0] + N[1] + N[2] ≤ 1000 )  \_\_\_\_\_mDisk : 각 기둥의 원판 크기 (1 ≤ mDisk[][] ≤ 1,000,000,000) |
| **void destroy()** |
| 각 테스트 케이스의 마지막에 호출된다.  빈 함수로 두어도 채점에는 영향을 주지 않는다. |
| **void go(int k, int mTop[3])** |
| 현재 상태에서 기둥 C로 모든 원판이 이동하기 위하여, 원판을 k 번 이동 후에 각 기둥에 가장 위에 있는 원판의 크기를 mTop 에 저장한다.  mTop[0] 에 기둥 A의 가장 위에 있는 원판의 크기를, mTop[1] 에 기둥 B의 가장 위에 있는 원판의 크기를, mTop[2] 에 기둥 C의 가장 위에 있는 원판의 크기를 넣는다.  만약 해당 기둥에 원판이 없는 경우 0을 저장한다.  기둥 C로 모두 옮겨진 이후에는 더 이상 이동하지 않는다.    ***\_\_Parameters***  \_\_\_\_\_k : 원판의 이동 횟수 (1 ≤ k ≤ 10)  \_\_\_\_\_mTop : 이동 후 가장 위에 있는 원판의 크기 |

**[예제]**

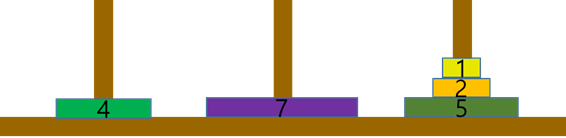
아래와 같이 함수 호출이 되는 경우를 살펴 보자.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Function | Description | mTop |
| 1 | init({1,2,2}, {4}, {1,7}, {2,5}) | {4} {1,7} {2,5} | - |
| 2 | go(1, ...) | {4} {7} {1,2,5} | 4, 7, 1 |
| 3 | go(1, ...) | {} {4,7} {1,2,5} | 0, 4, 1 |
| 4 | go(2, ...) | {1} {4,7} {2,5} **{1} {2,**4,7**} {5}** | 1, 2, 5 |
| 5 | go(2, ...) | {} {1,2,4,7} {5} **{5} {1,**2,4,7**} {}** | 5, 1, 0 |
| 6 | go(2, ...) | {1,5} {2,4,7} {} **{1,**5**} {4,**7**} {2}** | 1, 4, 2 |

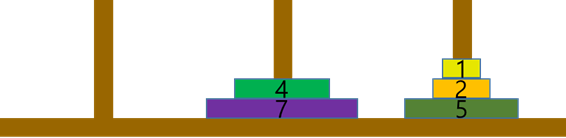
**[# 1]**



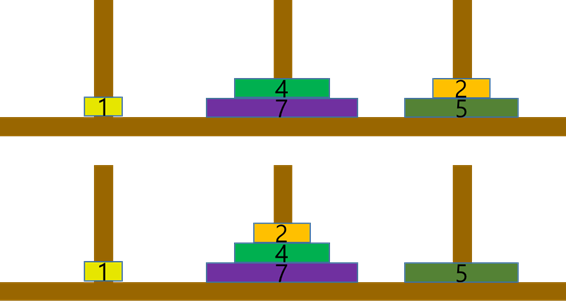
**[# 2]**



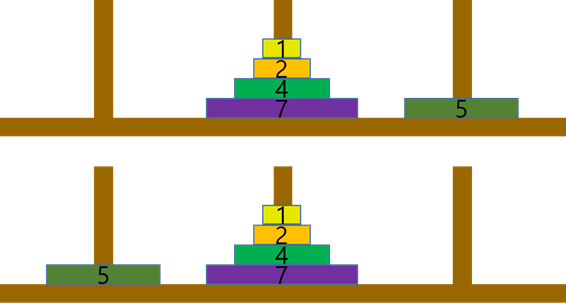
**[# 3]**



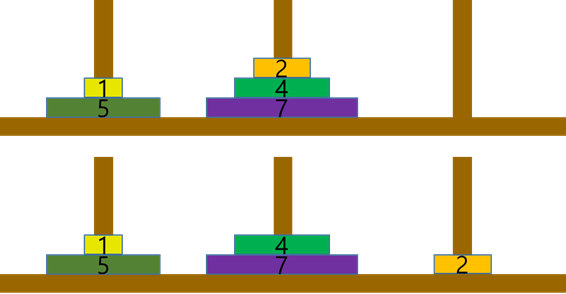
**[# 4]**



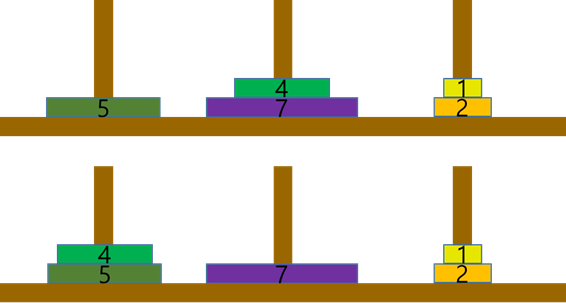
**[# 5]**



**[# 6]**



**[# ---]**



텍스트이(가) 표시된 사진

자동 생성된 설명

**[제약사항]**

1. 각 테스트 케이스 시작 시 init() 함수가, 종료 시 destroy() 함수가 호출된다.

2. 각 테스트 케이스에서 go() 함수는 최대 50,000회 호출된다.

**[입출력]**

입출력은 제공되는 Main 부분의 코드에서 처리하므로 User Code 부분의 코드에서는 별도로 입출력을 처리하지 않는다.

Sample input 에 대한 정답 출력 결과는 “#TC번호 결과” 의 포맷으로 보여지며 결과가 100 일 경우 정답, 0 일 경우 오답을 의미한다.