**[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]\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

모든 도시를 직선으로 연결한 직선 왕국이 만들어졌다.

도시는 N개가 있으며, 도시의 ID는 중복없이 0 ~ N - 1 로 부여된다.

수도는 ID 는 0 이며, 수도에 가까울수록 낮은 번호의 ID가 부여된다.

[Fig.1] 은 N이 5일 때의 직선 왕국의 도시의 구성을 나타낸 예이다.



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

모든 도시에서 파란색으로 표시된 수도로 이동하는 경로는 유일하다.

도시 1번에서 도시 N-1번의 각 도시에는 창고가 있다.

세금으로 납부된 곡물은 수도로 이동되기 전 각 도시 창고에 임시 보관된다.

직선 왕국에서 세금의 납부는 곡물을 바치라는 명령으로 시작된다. (order)

명령은 세금을 납부해야 하는 도시 A와 세금을 보낼 도시 B, 세금의 양을 지정해서 내려오며 아래와 같은 일이 일어난다.

\_\_\_\_\_1.  A 도시에서 세금의 양 만큼의 곡물을 즉시 수레에 싣고,

\_\_\_\_\_\_.  B 도시의 창고로 이동한다. 수레에 싣는 시간의 소요는 없다.

\_\_\_\_\_\_.  도시와 인접 도시와의 이동은 1시각이 소요된다.

\_\_\_\_\_2.  B 도시에 도착 후 B 도시에 있는 창고에 수레에 있는 모든 곡물을 쌓는다.

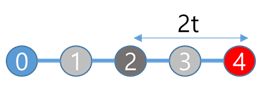
\_\_\_\_\_\_.  쌓는 시간의 소요는 없다.

각 도시에는 이동을 위한 수레와 인력은 항상 충분하다.

예를 들어 **[Fig. 2]**에서 세금을 납부하는 도시가 4번 도시이고 세금을 보낼 도시가 2번 도시라 할 때,

4번 도시에서 2번 도시까지의 거리는 2시각 거리이므로, 이동하는데, 2 시각이 소요된다.

t 시각에 명령이 내려오면 t + 2 시각에 2번 도시의 창고에 곡물이 도착한다.



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

세금을 납부해야 하는 도시와 세금을 보낼 도시는 수도를 제외한 모든 도시가 될 수 있다.

각 도시에 있는 창고에서 수도로 곡물의 이동은 왕실 소속의 담당 관리들이 담당한다.

왕실 소속의 담당 관리들은 M명이다.

관리의 파견은 별도의 함수 호출 없이 이루어진다.

왕실에서 대기하는 관리들은 매 시각 아래와 같은 일을 한다.

\_\_\_\_\_1.  **해당 시각에 이동 완료 예정인 곡물이 있다면, 이동이 완료되길 기다린다.**

\_\_\_\_\_2.  남아 있는 관리가 있다면, 3 ~ 5 를 반복한다.

\_\_\_\_\_3.  이동시간을 고려한 도착시간에 모든 창고의 곡물이 0 이 예상되거나, 파견할 도시가 없다면 대기한다.

\_\_\_\_\_4.  모든 도시와 인접한 도시의 이동은 1 시각으로 가정하여,

\_\_\_\_\_\_.  각 도시의 창고 중 관리가 현재 출발해서 도착 시각을 기준으로 가장 곡물이 많을 것으로 예상되는 도시의 창고를 선택한다.

\_\_\_\_\_\_.  이때, 이미 관리가 파견중인 도시는 배제된다.

\_\_\_\_\_\_.  만약 예상되는 곡물의 양이 같은 창고가 있다면, 도시 ID가 작은 창고가 우선시 된다.

\_\_\_\_\_5.  해당 도시의 창고로 관리 1명이 파견된다.

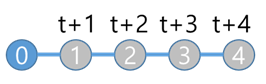
**[Fig. 3]**은 수도와 각 도시의 거리를 보여주며, 각 도시와의 거리는 관리 파견 시 고려해야 한다.

예를 들어 2번 도시의 경우 수도에서 2 시각의 거리에 있으므로, t 시각에 관리가 출발하면 t + 2시각에 도착하므로

t + 2 시에 도시 2번의 창고에 도착이 완료되는 곡물의 양을 기준으로 우선 순위를 고려해야 한다.

t 시각에 관리 파견을 결정할 때, 1번 도시는 t+1 시각에 예상되는 곡물의 양을, 2번 도시는 t+2 시각에 예상되는 곡물의 양을,

3번 도시는 t+3 시각에 예상되는 곡물의 양을, 4번 도시는 t+4 시각에 예상되는 곡물의 양을 비교하여 결정한다.



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [Fig. 3]**

파견된 관리는 아래의 일을 수행한다.

\_\_\_\_\_1.  파견된 도시의 창고까지 이동한다. 이동시 도시와 인접한 도시간의 이동은 1 시각 소요된다.

\_\_\_\_\_2.  파견된 도시의 창고의 모든 곡물을 수레에 싣는다. 싣는 소요 시간은 없다.

\_\_\_\_\_\_.  만약 같은 시각에 도착 예정인 곡물이 있다면, 도착이 완료된 후에 수레에 싣는다.

\_\_\_\_\_3.  수도 까지 이동한다. 이동시 도시와 인접한 도시간의 이동은 1 시각 소요된다.

\_\_\_\_\_4.  수레에 있는 모든 곡물을 수도에 옮겨 쌓는다. 옮겨 쌓는 시간 소요는 없다.

\_\_\_\_\_5.  임무를 마치고 즉시 새로운 임무를 받기 위해 대기한다.

\_\_\_\_\_\_.  파견됐던 도시도 새로운 관리가 파견될 수 있는 상태가 된다.

파견된 관리는 임무를 완료할 때까지 임무가 변경되는 일은 없다.

관리가 파견된 도시까지 이동하는 동안에도 해당 도시의 창고에 예상했던 곡식보다 더 많은 곡식이 추가 되는 경우도 존재한다.

이 경우에도 관리가 도착한 시각에 파견된 도시의 창고에 있는 모든 곡물을 수레에 싣는다.

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, int M)** |
| 각 테스트 케이스의 처음에 호출된다.  도시는 N 개가 있으며, 도시의 ID는 중복없이 0 ~ N - 1 로 부여된다.  수도는 ID 는 0 이며, 수도에 가까울수록 낮은 번호의 ID가 부여된다.  현재 시각은 0 이다.  모든 도시의 창고는 비어있고, 왕실 소속 관리는 수도에 M 명이 대기하고 있다.  왕실 소속 관리는 현재 시점에 이동해도 예상되는 곡물이 없으므로 대기하고 있다.    ***\_\_Parameters***  \_\_\_\_\_N : 도시의 개수 ( 10 ≤ N ≤ 200 )  \_\_\_\_\_M : 왕실 소속 관리의 수 ( 1 ≤ M ≤ 50 ) |
| **void destroy()** |
| 각 테스트 케이스의 마지막에 호출된다.  빈 함수로 두어도 채점에는 영향을 주지 않는다. |
| **int order(int tStamp, int mCityA, int mCityB, int mTax)** |
| tStamp 시각에 도시 mCityA 에 mTax 만큼의 곡물을 도시 mCityB 로 납부하라는 명령을 내린다.  도시 mCityA 에서 곡물 mTax 만큼을 도시 mCityB 로 이동 시킨다.  이동시 인접한 도시마다 1 시각의 시간이 소요된다.  곡물을 싣거나 쌓을 때는 시간의 소요가 없다.  mCityA ≠ mCityB 임이 보장된다.  tStamp 시각 현재 수도에 있는 곡물의 양을 반환한다.  세금 납부시 도시 mCityA 창고에 있는 곡물을 빼서 옮기는 것이 아님을 주의하라.  tStamp 시각에 수도에 도착 예정인 곡물이 있다면, 해당 곡물의 이동이 완료된 후의 양을 반환한다.    ***\_\_Parameters***  \_\_\_\_\_tStamp : 명령을 내린 시각 (1 ≤ tStamp ≤ 100,000,000)  \_\_\_\_\_mCityA : 세금을 납부하는 도시 ( 1 ≤ mCityA < N )  \_\_\_\_\_mCityB : 세금을 보낼 도시 ( 1 ≤ mCityB < N )  \_\_\_\_\_mTax : 이동할 곡물의 양 ( 1 ≤ mTax ≤ 10 )    ***\_\_Results***  \_\_\_\_\_수도에 있는 곡물의 양 |
| **int check(int tStamp)** |
| tStamp 시각 현재 수도에 있는 곡물의 양을 반환한다.  tStamp 시각에 수도에 도착 예정인 곡물이 있다면, 해당 곡물의 이동이 완료된 후의 양을 반환한다.    ***\_\_Parameters***  \_\_\_\_\_tStamp : 시각 (1 ≤ tStamp ≤ 100,000,000)  ***\_\_Results***  \_\_\_\_\_수도에 있는 곡물의 양 |

**[예제]**

아래와 같이 함수 호출이 되는 경우를 살펴 보자.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Function** | **Description** | **Return** |
| 1 | init(10, 2) | 수도를 포함한 도시는 10개이다. 왕실소속관리 2명이 대기하고 있다. 현재 시각은 0이다. 모든 도시의 창고에 있는 곡물의 양은 0 이다. 수도에 있는 곡물의 양은 0 이다. |  |
| 2 | order(2, 2, 9, 9) | 시각 2 에 도시 2번에 세금 9를 도시 9로 납부하라는 명령이 내려진다. | 0 |
| 3 | order(4, 5, 7, 7) | 시각 4 에 도시 5번에 세금 7을 도시 7로 납부하라는 명령이 내려진다. | 0 |
| 4 | order(5, 1, 5, 8) | 시각 5 에 도시 1번에 세금 8을 도시 5로 납부하라는 명령이 내려진다. | 0 |
| 5 | order(7, 1, 7, 10) | 시각 7 에 도시 1번에 세금 10을 도시 7로 납부하라는 명령이 내려진다. | 0 |
| 6 | order(9, 5, 4, 5) | 시각 9 에 도시 5번에 세금 5를 도시 4로 납부하라는 명령이 내려진다. | 0 |
| 7 | check(10) | 시각 10 현재 수도의 곡물의 양을 반환한다. | 0 |
| 8 | check(12) | 시각 12 현재 수도의 곡물의 양을 반환한다. | 0 |
| 9 | order(14, 2, 4, 9) | 시각 14 에 도시 2번에 세금 9를 도시 4로 납부하라는 명령이 내려진다. | 0 |
| 10 | order(15, 2, 5, 2) | 시각 15 에 도시 2번에 세금 2를 도시 5로 납부하라는 명령이 내려진다. | 0 |
| 11 | check(17) | 시각 17 현재 수도의 곡물의 양을 반환한다. | 0 |
| 12 | check(19) | 시각 19 현재 수도의 곡물의 양을 반환한다. | 7 |
| 13 | check(21) | 시각 21 현재 수도의 곡물의 양을 반환한다. | 16 |
| 14 | check(23) | 시각 23 현재 수도의 곡물의 양을 반환한다. | 16 |
| 15 | check(24) | 시각 24 현재 수도의 곡물의 양을 반환한다. | 16 |
| 16 | check(26) | 시각 26 현재 수도의 곡물의 양을 반환한다. | 30 |
| 17 | order(27, 1, 6, 10) | 시각 27 에 도시 1번에 세금 10을 도시 6으로 납부하라는 명령이 내려진다. | 30 |
| 18 | check(29) | 시각 29 현재 수도의 곡물의 양을 반환한다. | 30 |
| 19 | check(31) | 시각 31 현재 수도의 곡물의 양을 반환한다. | 40 |
| 20 | order(32, 2, 3, 8) | 시각 32 에 도시 2번에 세금 8을 도시 3으로 납부하라는 명령이 내려진다. | 40 |
| 21 | order(33, 5, 3, 3) | 시각 33 에 도시 5번에 세금 3을 도시 3으로 납부하라는 명령이 내려진다. | 40 |

아래 표에서 예상되는 곡물의 양은 관리의 파견 여부를 고려하지 않은 수도에서 출발 했을 때, 기대되는 곡물의 양을 나타낸다.

표에서 음영표시는 관리가 파견중인 도시를 나타낸다.

**[#1]**

순서 1에서 init(10, 2) 함수 실행 후 모든 도시의 창고의 상태는 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

현재 시각은 0이며, 2명의 관리가 수도에 대기하고 있다.

현재 시각에 관리가 출발하여 각 도시에 도착 했을 때, 예상되는 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

이동해도 얻을 수 있는 곡물의 양이 없으므로 관리들은 수도에서 대기한다.

**[#2]**

순서 2에서 order(2, 2, 9, 9) 함수 실행 후 모든 도시의 창고의 상태는 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

도시 2번에서 곡물 9가 도시 9로 출발한다.

2번 도시에서 9번 도시까지의 거리는 7이므로, 현재시각에서 7시각 이후인 시각 9에 9번 도시의 창고에 9만큼의 곡물이 추가될 예정이다.

현재 시각에 관리가 출발하여 각 도시에 도착 했을 때, 예상되는 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 |

수도에 관리가 대기하고 있고, 도시 9번이 현재 관리가 파견된 도시가 아니므로, 도시 9번에 관리 1명를 파견한다.

파견된 관리는 현재 시각인 2시각에 출발하여,

도시 9번까지 9시각 이동하여, 시각 11에 도시 9번에 도착 할 예정이고,

도착 시 해당 도시 창고에 있는 곡물을 모두 싣고,

다시 9시각 이동하여 20시각에 수도에 곡물을 쌓고, 임무를 완료할 예정이다.

**[#3]**

순서 3에서 order(4, 5, 7, 7) 함수 실행 후 모든 도시의 창고의 상태는 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

도시 5번에서 도시 7까지의 거리는 2이므로, 시각 6에 7번 도시의 창고에 곡물 7이 추가될 예정이다.

현재 시각에 수도에서 관리가 출발하여 각 도시에 도착 했을 때, 예상되는 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 9 |

도시 9번이 예상되는 곡물이 가장 많지만, 도시 9번에는 이미 관리가 파견 중이므로, 도시 7번에 관리를 파견한다.

\_\_※ 9번 도시의 예상 곡물의 양은 관리가 파견중이므로 현재 시점에서 0이 정확한 값이지만, 파견중인 도시는 관리의 파견에서 배제되므로 고려대상이 아니다.

\_\_\_\_표에서는 관리 파견을 고려하지 않은 곡물의 양이기 때문에 9번 도시의 예상 곡물의 양은 9로 표기하였다.

**[#4]**

순서 4에서 order(5, 1, 5, 8) 함수 실행 후 모든 도시의 창고의 상태는 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

현재 시각에 수도에서 관리가 출발하여 각 도시에 도착 했을 때, 예상되는 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 7 | 0 | 9 |

도시 5번에도 현재 시각에 출발하여 얻을 수 있는 곡물이 있지만, 대기중인 관리가 없다.

**[-----]**

시각 6 에 도시 7번의 창고에 곡물 7이 쌓인다.

**[#5]**

순서 5에서 order(7, 1, 7, 10) 함수 실행 후 모든 도시의 창고의 상태는 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |

현재 시각에 수도에서 관리가 출발하여 각 도시에 도착 했을 때, 예상되는 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 17 | 0 | 9 |

**[#6]**

시각 9에 도시 9번 창고에 곡물 9, 도시 5번 창고에 곡물 8이 쌓인다.

순서 6에서 order(9, 5, 4, 5) 함수 실행 후 모든 도시의 창고의 상태 및 현재 시각 관리가 이동하여 얻을 수 있는 예상 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 7 | 0 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 17 | 0 | 9 |

**[#7]**

시각 10에 도시 4번 창고에 곡물 5가 쌓인다.

시각 10에 check(10) 함수의 실행 결과는 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 7 | 0 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 17 | 0 | 9 |

수도에 곡물이 없으므로 0을 반환한다.

**[-----]**

시각 11에 파견된 두 관리가 각각 도시 7번, 도시 9번에 도착한다.

해당 도시의 곡물을 모두 수레에 싣고, 수도로 출발한다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 10 | 0 | 0 |

**[#8]**

시각 12에 check(12) 함수의 실행 결과 수도에 곡물이 없으므로 0을 반환한다.

**[-----]**

시각 13에 도시 7번 창고에 곡물 10이 도착하여 쌓인다.

**[#9]**

순서 9에서 order(14, 2, 4, 9) 함수 실행 후 모든 도시의 창고의 상태 및 현재 시각 관리가 이동하여 얻을 수 있는 예상 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 10 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 14 | 8 | 0 | 10 | 0 | 0 |

**[#10]**

순서 10에서 order(15, 2, 5, 2) 함수 실행 후 모든 도시의 창고의 상태 및 현재 시각 관리가 이동하여 얻을 수 있는 예상 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 0 | 0 | 0 | 0 | 5 | 8 | 0 | 10 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 14 | 10 | 0 | 10 | 0 | 0 |

**[-----]**

시각 16에 도시 4번에 곡물 9가 도착하여 창고에 쌓인다.

**[#11]**

순서 11에서 check(17) 의 실행 결과는 수도에 있는 곡물의 양이 0이므로 0을 반환한다.

**[-----]**

시각 18에 도시 5번에 곡물 2가 도착하여 창고에 쌓인다.

시각 18에 관리가 수도로 돌아온다. 도시 7번에서 가져온 곡물 7을 수도에 쌓는다.

도시 7번은 관리를 파견할 수 있는 상태가 된다.

관리는 수도에서 출발하면 도시 4번에 도착해서 곡물 14를 얻을 수 있으므로, 도시 4로 출발한다.

**[#12]**

순서 12에서 check(19) 의 실행 결과는 수도에 있는 곡물의 양이 7 이므로 7을 반환한다.

**[-----]**

시각 20에 관리가 수도로 돌아온다. 도시 9번에서 가져온 곡물 9를 수도에 쌓는다. 수도의 곡물은 16이 된다.

도시 9번은 관리를 파견할 수 있는 상태가 된다.

관리는 수도에서 출발하면 도시 5번에 도착해서 곡물 10를 얻을 수 있으므로, 도시 5 로 출발한다.

**[#13]**

순서 13에서 check(21) 의 실행 결과는 수도에 있는 곡물의 양이 16 이므로 16을 반환한다.

**[-----]**

시각 22에 관리가 도시 4에 도착한다. 도시 4의 곡물 14를 싣고, 수도로 출발한다.

**[#14]**

순서 14에서 check(23) 의 실행 결과는 수도에 있는 곡물의 양이 16 이므로 16을 반환한다.

**[#15]**

순서 15에서 check(24) 의 실행 결과는 수도에 있는 곡물의 양이 16 이므로 16을 반환한다.

**[-----]**

시각 25에 관리가 도시 5에 도착한다. 도시 5의 곡물 10를 싣고, 수도로 출발한다.

**[#16]**

시각 26에 관리가 수도로 돌아온다. 도시 4로부터 가져온 곡물 14를 수도에 쌓는다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |

현재 시각에 출발하면 도시 7에서 곡물 10을 얻을 수 있으므로, 관리는 도시 7번으로 임무를 받고 출발한다.

순서 16에서 check(26) 의 실행 결과는 수도에 있는 곡물의 양이 30 이므로 30을 반환한다.

**[#17]**

순서 17에서 order(27, 1, 6, 10)함수 실행 후 모든 도시의 창고의 상태 및 현재 시각 관리가 이동하여 얻을 수 있는 예상 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 0 |

**[#18]**

순서 18에서 check(29) 의 실행 결과는 수도에 있는 곡물의 양이 30 이므로 30을 반환한다.

**[-----]**

시각 30에 관리가 수도로 도착한다. 도시 5에서 가져온 곡물 10을 수도에 쌓는다.

수도의 곡물은 40이 된다.

수도에서 출발하여 도시 6에 도착하면 곡물 10을 얻을 수 있으므로, 관리는 도시 6번으로 출발한다.

**[#19]**

순서 19에서 check(31) 의 실행 결과는 수도에 있는 곡물의 양이 40 이므로 40을 반환한다.

**[#20]**

시각 32에 도시 6번 창고에 곡물 10이 도착하여 쌓인다.

순서 20에서 order(32, 2, 3, 8)함수 실행 후 모든 도시의 창고의 상태 및 현재 시각 관리가 이동하여 얻을 수 있는 예상 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 40 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 8 | 0 | 0 | 10 | 10 | 0 | 0 |

**[#21]**

시각 33에 도시 3번 창고에 곡물 8이 도착하여 쌓인다.

시각 33에 관리가 도시 7에 도착한다. 도시 7번에 있는 곡물 10을 가지고, 수도로 출발한다.

순서 21에서 order(33, 5, 3, 3)함수 실행 후 모든 도시의 창고의 상태 및 현재 시각 관리가 이동하여 얻을 수 있는 예상 곡물의 양은 아래와 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| storage | 40 | 0 | 0 | 8 | 0 | 0 | 10 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| expected | 0 | 0 | 0 | 11 | 0 | 0 | 10 | 0 | 0 | 0 |

**[제약사항]**

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

2. 각 테스트 케이스에서 order() 함수는 최대 10,000회 호출된다.

3. 각 테스트 케이스에서 check() 함수는 최대 10,000회 호출된다.

4. 각 테스트 케이스에서 API 호출시 tStamp 의 값은 중복되지 않으며, 증가한다.

**[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.