2018
서울대학교 프로그래밍 경시대회

Division 2

주최 및 주관

후원

2018년 9월 9일

참가자를 위한 도움말

주의 사항

- 대회 시간은 13:00부터 17:00까지입니다. 대회가 진행되는 동안 **인터넷 검색 및 전자기기 사용 등을 하실 수 없습니다.** 단, 아래의 문서에 한해 대회 진행 중에도 참고하실 수 있으며, 책과 노트 등을 가져오신 경우 역시 참고하실 수 있습니다.
  - C/C++ reference: https://en.cppreference.com/w/
  - Python 2 reference: https://docs.python.org/2/
  - Python 3 reference: https://docs.python.org/3.6/
  - Java documentation: https://docs.oracle.com/javase/8/docs/api/
- 대회는 Baekjoon Online Judge(https://www.acmicpc.net/) 플랫폼을 이용하여 진행됩니다. 별도로 제공되는 계정 정보를 이용하여 로그인하신 뒤 코드 제출 및 결과 확인 등을 하실 수 있습니다.
- 제출하실 답안 코드는 C, C11, C++, C++11, C++14, C++17, Java, Python 2, Python 3, PyPy2, PyPy3 **로만** 작성하셔야 합니다. 단, 출제자는 C++14와 C++17을 제외한 다른 언어로 주어진 시간 제한과 메모리 제한을 지키며 올바른 답을 내는 코드를 작성할 수 있다는 보장을 하지 않습니다.
- 모든 입력은 **표준 입력**으로 주어지며, 모든 출력은 **표준 출력**으로 합니다.
- 테스트 케이스가 존재하는 문제의 경우, 테스트 케이스에 대한 출력을 모아서 하실 필요 없이, 각 테스트 케이스를 처리할 때마다 출력해도 괜찮습니다.
- **중요!! 리턴 코드**와 **표준 오류**(standard error, stderr) **스트림 출력**에 주의하십시오. 프로세스가 0이 아닌 리턴 코드를 되돌리는 경우나 **표준 오류 스트림에 출력**을 하는 경우 "**런타임 에러**"를 받게 됩니다.
- 문제에 대한 질의 사항은 대회 페이지의 질문 기능을 사용해 주시기 바랍니다. 이 때 대답해 드리기 어려운 질문에 대해서는 "**답변을 드릴 수 없습니다**"로 대답될 수 있으므로 유의하십시오.
- 이 문제지는 참고용입니다. 문제지에 적힌 문제와 플랫폼에 올라온 문제가 다를 경우, 플랫폼의 문제가 우선합니다. 특히 색이 들어간 이미지는 플랫폼에서 보시기 바랍니다.

채점 결과에 대하여

맞았습니다!! 제출하신 답안이 모든 테스트 데이터를 정해진 시간 안에 통과하여 정답으로 인정되었음을 의미합니다.

틀렸습니다 제출하신 답안 프로그램이 테스트 데이터에 대해 생성한 출력이 출제자의 정답과 일치하지 않음을 의미합니다.

컴파일 에러 제출하신 답안 프로그램을 컴파일하는 도중 오류가 발생하였음을 의미합니다.

런타임 에러 제출하신 답안 프로그램을 실행하는 도중 프로세스가 비정상적으로 종료되었음을 의미합니다.

시간 초과 제출하신 답안 프로그램이 정해진 시간 안에 종료되지 않았음을 의미합니다.

출력 형식이 잘못되었습니다 답을 올바르게 구했으나, 답안 프로그램의 출력 형식이 문제에 나와 있는 출력 형식과 다름을 의미합니다. 줄 뒤에 의미 없는 공백을 두 칸 이상 출력하거나 의미 없는 빈 줄을 출력할 경우 이 결과를 받을 수 있습니다.

출력 초과 답안 프로그램이 정답에 비해 너무 많은 출력을 했음을 의미합니다. 이 결과는 틀렸습니다와 같은 의미를 갖습니다.

만약 여러 가지의 원인으로 인해 "**맞았습니다!!**"가 아닌 다른 결과를 얻으셨다면, 그 중 어떤 것도 결과가 될 수 있습니다. 예를 들어 답도 잘못되었고 비정상적인 동작도 수행하는 코드를 제출하신 경우 대부분 "**런타임 에러**"를 받으시게 되지만, 경우에 따라서 "**틀렸습니다**"를 받을 수도 있습니다.

문제별 시간 제한은 다음과 같습니다.

- A : 1.0 s
- B : 1.0 s
- C : 1.0 s
- D : 2.0 s
- E : 1.0 s
- F : 1.0 s
- G : 1.0 s
- H : 1.0 s
- I : 0.5 s
- J : 2.0 s

문제별 메모리 제한은 모든 문제에 대해 512MB로 동일합니다.

A. 5차 전직

메이플스토리 뉴비 키파가 드디어 레벨 200을 달성하고 **5차 전직**이라는 시스템을 이용해 캐릭터를 더욱 강력하게 만들려고 합니다. 5차 전직을 하려면 먼저 퀘스트를 통해 **아케인스톤**이라는 아이템을 받아야 합니다. 아케인스톤을 활성화시키면 캐릭터가 얻는 경험치를 아케인스톤에 모을 수 있습니다. 5차 전직을 하기 위해서는 총 n개의 퀘스트를 진행해서 n개의 아케인스톤을 받아야 하며, 각각의 아케인스톤에 5억 이상의 경험치를 모으면 5차 전직을 진행할 수 있는 자격이 주어집니다.

i번째 퀘스트를 진행하면 a_i의 경험치와 i번째 아케인스톤이 주어집니다. 퀘스트로 얻는 경험치도 사냥으로 얻는 것과 똑같은 **경험치**이기 때문에, i번째 퀘스트의 보상 경험치를 받을 때 활성화되어 있던 아케인스톤에는 a_i의 경험치가 추가됩니다.

메이플월드의 아케인스톤입니다. 멋지죠.

원래 메이플스토리에서는 한 번에 하나의 아케인스톤만 활성화시켜 놓을 수 있고, 각각의 아케인스톤에는 최대 5억의 경험치를 채울 수 있습니다. 그러나 해킹에는 자신이 있었던 메린이 키파는 서버를 해킹해 **아케인스톤의 최대 경험치 제한을 없애 버리고**, 최대 k개의 아케인스톤이 동시에 활성화되어 있을 수 있도록 바꿨습니다. 따라서 한 퀘스트의 보상 경험치가 여러 개의 아케인스톤에 추가될 수 있습니다. 예를 들어 1번째와 3번째 아케인스톤이 활성화되어 있는 상태에서 2번째 퀘스트를 진행해 $100,000$의 경험치와 2번째 아케인스톤을 획득하면, 1번째와 3번째 아케인스톤에 각각 $100,000$의 경험치가 추가되고 2번째 아케인스톤은 모인 경험치가 0인 상태로 받게 됩니다.

키파는 퀘스트를 원하는 순서대로 진행할 수 있지만, 같은 퀘스트를 두 번 이상 진행할 수는 없습니다. 키파는 퀘스트를 진행하면서 아케인스톤을 적절히 활성화 또는 비활성화시켜서 아케인스톤에 모인 경험치의 합을 최대화하고 싶습니다. 모인 경험치의 합이 커지면 어쨌든 기분이 좋으니까요. 키파를 대신해서 이 값을 구해 주세요!

입력

첫째 줄에 정수 n과 $k(1 \le k \le n \le 3 \cdot 10^5)$가 주어집니다.

둘째 줄에 n개의 정수가 공백을 사이에 두고 주어집니다. i번째 정수는 a_i이며 0보다 크고 10^8보다 작거나 같습니다.

출력

첫째 줄에 키파가 아케인스톤에 모을 수 있는 경험치의 합의 최댓값을 출력합니다.

예제 입출력

standard input	standard output
3 2 100 300 200	800

노트

먼저 첫 번째 퀘스트를 진행하고 첫 번째 아케인스톤을 받은 뒤 활성화시킵니다. 그 다음 세 번째 퀘스트를 진행하고 세 번째 아케인스톤을 받은 뒤 활성화시킵니다. 마지막으로 두 번째 퀘스트를 진행합니다.

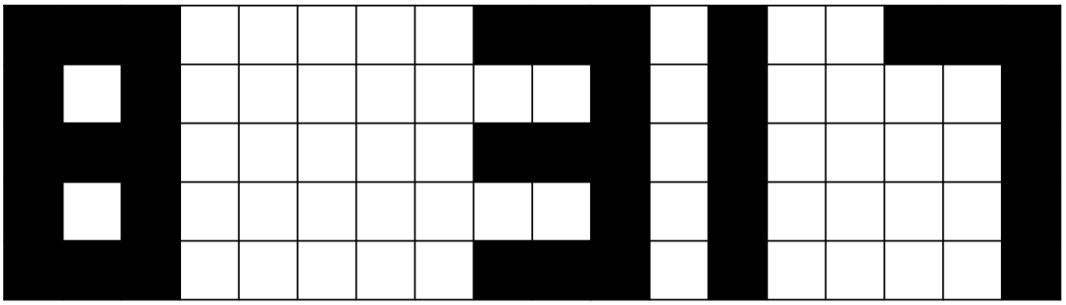
이 경우 첫 번째 아케인스톤에 500, 세 번째 아케인스톤에 300의 경험치가 모여 합이 800이 되고, 이보다 모인 경험치의 합을 크게 할 수는 없습니다.

B. 시그널

zxcvber는 외계인을 연구하는 과학자다. 그는 지난 10년간 우주에서 오는 시그널를 연구했지만, 아무런 성과가 없었다. 그러던 어느 날, 갑자기 우주에서 이상한 시그널이 오기 시작했다. zxcvber는 매우 기뻐하며 시그널을 받아서 분석해보았다. 시그널은 0과 1로 이루어져 있는데, 여기서는 편의상 0을 '.', 1을 '#'으로 표시한다. 시그널은 다음과 같았다.

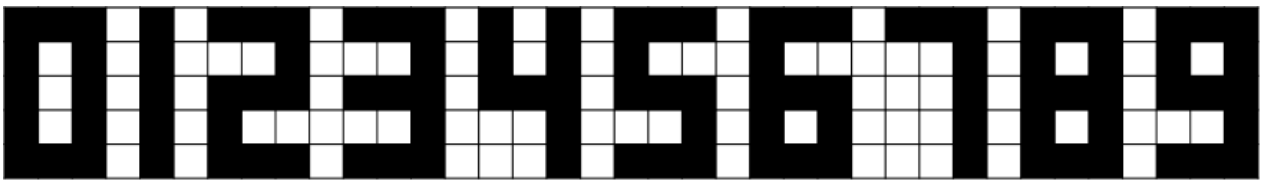
```
###.....###.#..####.#.......#.#....####.....###.#....##.#.......#.#....####.....###.#....#
```

다른 여러 시그널들을 분석해본 결과, zxcvber는 시그널의 길이가 항상 5의 배수라는 것을 알게 되었다. 시그널을 다섯 개로 쪼개면 뭔가 규칙이 보이지 않을까 생각한 zxcvber는 시그널을 같은 길이의 5개의 시그널로 쪼갰다. 그러자 놀라운 일이 벌어졌다.('#'을 검은색, '.'을 흰색으로 표시했다.)



시그널은 디지털 숫자를 나타내고 있었다! 1-3열에 8, 9-11열에 3, 13열에 1, 그리고 16-18열에 7이 나타난 것이다. 이 숫자들이 특별한 의미를 갖고 있을 것이라 생각한 zxcvber는 다른 시그널들도 해독을 하기 시작했다. 하지만 시그널들의 길이가 너무 길어서, 일일히 손으로 확인하기에는 한계가 있었다. 다만, 짧은 시그널들을 분석하면서 zxcvber는 시그널의 규칙들을 파악할 수 있었다.

1. 시그널은 '.'과 '#'으로 이루어져 있다.
2. 시그널을 해독한 결과에는 반드시 숫자가 1개 이상 있다.
3. 시그널에 등장하는 모든 '#'은 올바른 숫자 패턴에 포함되어 있다.
4. 숫자와 숫자 사이에는 1열 이상의 공백이 있다. 여기서 공백은, 열의 성분이 모두 '.'인 열을 의미한다.
5. 0부터 9는 아래와 같이 나타난다.(역시 '#'을 검은색, '.'을 흰색으로 표시했다.)



주의할 점은, 1은 다른 숫자들과는 다르게 1열을 차지한다는 것이다. zxcvber를 도와 시그널을 해독해보자!

입력

입력의 첫 줄에는 시그널의 길이 N이 주어진다. ($5 \le N \le 100,000$, N은 5의 배수)

다음 줄에 시그널이 주어진다. zxcvber가 찾아낸 규칙을 따르는 시그널만이 입력으로 주어진다.

출력

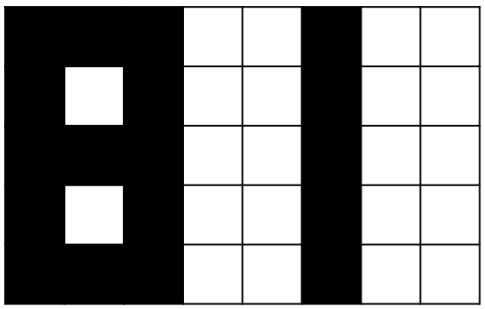
첫 번째 줄에 시그널을 해독하여 나오는 숫자들을 순서대로 출력한다.

예제 입출력

example input
40 ###..#..#.#..#..###..#..#.#..#..###..#..
example output
81

노트

예제 시그널을 8개씩 끊어서 읽으면 다음과 같다.



C. 화살표 연산자

NOTE: 이 문제의 내용은 C++11 이후의 표준을 따릅니다.

새내기들을 위해 C++ 스터디를 준비하던 키파는 인터넷에서 신기한 연산자를 발견했다. 바로 화살표 연산자(`<--`)이다.

프로그램	출력

`#include <iostream>`
 	
`int main(){`
`  int x = 10;`
 	
`  while( 0 <-- x ){`
`  std::cout << x << std::endl;`
`  }`
`  return 0;`
`}`
	9
	8
	7
	6
	5
	4
	3
	2
	1

위와 같이 화살표 연산자를 사용하면 연산자 오른쪽에 있는 변수가 연산자 왼쪽의 값을 향해 다가간다! 심지어 이 연산자는 화살표의 길이(`-`의 개수)를 늘려서 값이 바뀌는 속도를 더 빠르게 할 수도 있다.

프로그램	출력

`#include <iostream>`
 	
`int main(){`
`  int x = 10;`
 	
`  while( 0 <---- x ){`
`  std::cout << x << std::endl;`
`  }`
`  return 0;`
`}`
	8
	6
	4
	2

화살표의 길이가 2인 첫 번째 코드에서는 수가 9개 출력되었으나, 길이를 4로 늘린 두 번째 코드에서는 수가 4개밖에 출력되지 않았다. 변수가 2배 빠르게 값으로 다가간 것이다!

사실 이 화살표 연산자는 전위 감소 연산자(prefix decrement operator, `--x`)를 이용한 농담으로, `0 <-- x`은 `--x > 0`을 화살표처럼 보이도록 눈속임한 것이다. 따라서 첫 번째 코드의 반복문은 아래와 같이 동작한다.

1. `x`의 값을 1 감소시킨다.
2. 만약 `x`의 값이 0보다 크지 않다면 프로그램을 종료한다.
3. `x`의 값을 출력하고 처음으로 돌아간다.

감소 연산자는 아래와 같은 특징이 있다.

- `-`가 여러 개 연속되어 있을 때는 반드시 앞에서부터 두 개씩 끊어서 해석된다. 예를 들어 `------x`는 `-- -- -- -x` 또는 `- -- - --x` 등이 아니라 반드시 `-- -- --x`로 해석된다.
- 감소 연산자는 단항 부정 연산자(unary minus operator, `-x`)가 적용된 식에는 적용할 수 없다. 예를 들어 `-----x`는 `-- -- -x`로 해석되고, 감소 연산자가 `-x`에 적용되었으므로 컴파일에 실패한다.

두 번째 코드의 `0 <---- x`는 `-- --x > 0`과 같고, 감소 연산자가 두 번 적용되었으므로 `x`의 값이 한번에 2씩 줄어든다.

위의 코드에서 `x`의 초기값과 화살표의 길이를 바꿨을 때, 몇 개의 수가 출력될지 예상해 보자.

입력

첫 줄에 변수 `x`의 초기값을 뜻하는 정수 X와 화살표의 길이를 뜻하는 정수 N($-100 \le X \le 100$, $0 \le N \le 10$)이 주어진다.

출력

첫째 줄에 프로그램의 실행 결과를 출력한다.

- 만약 프로그램이 정상적으로 종료된다면 프로그램이 출력한 수의 개수를 출력한다.
- 만약 프로그램이 무한히 많은 수를 출력한다면 `INFINITE`를 출력한다.
- 만약 프로그램이 지문에 주어진 조건에 의해 컴파일에 실패할 경우 `ERROR`를 출력한다.

예제 입출력

standard input	standard output
`10 4`	`4`
`-5 5`	`ERROR`
`3 0`	`INFINITE`

노트

세 번째 예시는 `x`의 값이 항상 3이므로 프로그램이 3을 무한히 출력한다.

D. 팬이에요

상수는 진표의 열렬한 팬이다. 그는 팬심을 담아 진표에게 직접 만든 선풍기를 선물해 주었다. 선풍기를 돌리고 놀던 진표는 선풍기를 일정 각도 이상 돌리면 날개의 자취가 원 형태가 된다는 것을 발견했다.

선풍기의 회전 중심을 원점으로 하는 좌표평면 위에 선풍기의 날개를 놓았다고 가정하자. 이 때 선풍기의 날개는 원점을 포함하며 꼭짓점이 N 개인 단순다각형으로 나타낼 수 있다. 또한 날개는 안전을 위해 다음과 같은 조건을 만족하도록 디자인되었다.

선풍기 날개 내부에 있는 임의의 점 P에 대해, 원점 O와 P를 잇는 선분 위의 모든 점 역시 날개 내부에 있다.

이 때, 날개의 자취가 원이 되는 최소 회전 각도를 구해 보자. 선풍기는 반시계 방향으로 돌아간다.

입력

첫 줄에 선풍기 날개의 꼭짓점의 수 $N (3 \le N \le 300,000)$이 주어진다.

두 번째 줄부터 N 개의 줄에 걸쳐 각 꼭짓점의 좌표가 반시계 방향으로 순서대로 주어진다. 각 좌표는 x 좌표와 y 좌표를 의미하는 두 개의 정수로 이루어지며, 각 좌푯값의 절댓값은 10^6 이하이다.

이 다각형은 원점을 포함하며, 인접한 두 선분을 제외한 다른 선분끼리는 만나지 않는다. 또한 다각형에서 연속한 3개의 점이 한 직선 위에 있지 않음이 보장된다.

출력

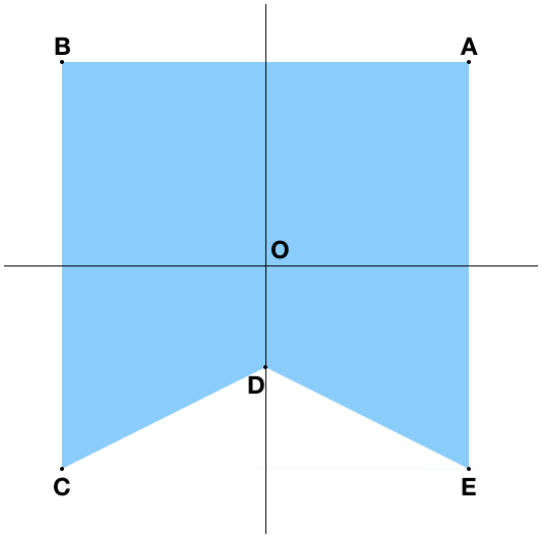
선풍기를 반시계 방향으로 θ만큼 돌렸을 때 날개의 자취가 원이 되는 최소 θ를 도(°) 단위로 구해 출력한다. 절대 또는 상대 오차가 10^{-6} 이하면 정답으로 처리된다.

예제 입출력

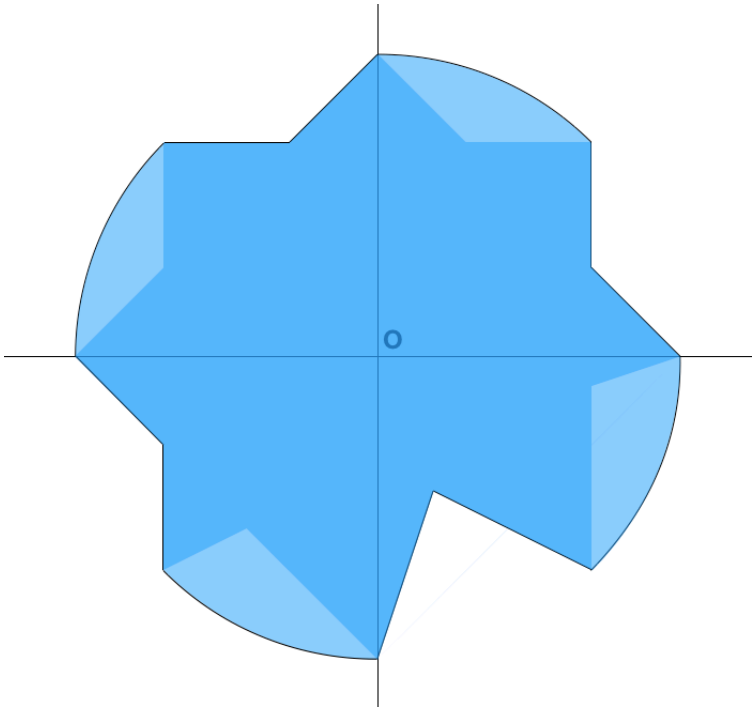
standard input	standard output
5 2 2 -2 2 -2 -2 0 -1 2 -2	90.000000
4 2 2 2 3 -2 -2 -2 -3	180.000000

노트

첫 번째 예제의 선풍기는 아래와 같다.



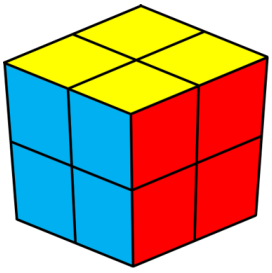
이를 반시계 방향으로 45° 돌렸을 때 자취는 아래와 같다. 처음 도형과 마지막 도형을 짙게 칠하였다.



이렇게 90°를 돌리는 순간 자취는 반지름이 $2\sqrt{2}$인 원이 된다. 따라서 답은 90이다.

E. 작은 큐브러버

작은 큐브러버는 작은 8개의 1×1×1 정육면체 조각을 갖고 있었어요. 각 조각에는 색이 칠해진 스티커가 붙어 있는 면이 세 개씩 있는데, 그 세 개의 면은 하나의 꼭짓점을 공유하고 있어요. 작은 큐브러버는 이 조각들을 잘 조립해서 각 면에 있는 4장의 스티커의 색이 서로 같고, 서로 다른 면에 있는 스티커는 서로 색이 다른 작은 2×2×2 큐브를 만들었어요. 작은 큐브러버는 작은 큐브를 매우 사랑했답니다.



그런데 어느 날 작은 큐베러버가 날아와 작은 큐브에 붙어 있는 스티커들을 뒤섞고, 작은 큐브를 다시 8개의 조각으로 분해해 버렸어요. 스티커들을 뒤섞었다는 것은 임의의 두 개의 스티커를 떼어낸 다음 위치를 바꿔서 붙이는 것을 충분히 많이 반복했다는 뜻이에요.

작은 큐브러버는 너무 슬펐지만 용기를 내어, 작은 큐베러버를 **삐**-하고 8개의 조각들을 되찾았어요. 작은 큐브러버는 이 조각들을 조립해서 다시 작은 큐브를 만들고 싶어해요. 작은 큐브러버를 도와주세요!

입력

8개의 줄에 걸쳐 각 조각을 스티커가 붙어 있는 세 개의 면이 전부 보이도록 놓았을 때, 각 스티커에 칠해진 색을 윗면, 왼쪽 면, 오른쪽 면 순서로 읽은 결과가 주어진다. 각각의 색은 `B`(blue), `G`(green), `O`(orange), `R`(red), `W`(white), `Y`(yellow) 중 하나의 알파벳으로 표현된다. 예를 들어 위의 이미지의 중앙에 있는 조각은 `Y B R`로 주어진다.

입력에는 각각의 알파벳이 같은 개수만큼 등장함이 보장된다.

출력

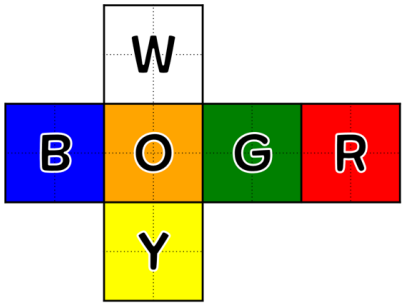
조각을 조립해서 문제의 조건을 만족하는 작은 2×2×2 큐브를 만들 수 있을 경우 `YES`, 불가능하다면 `NO`를 출력한다.

예제 입출력

standard input	standard output
W B O B W R G O Y R W G Y O B O G W B R Y G Y R	YES
R O W B Y G W B O W R G Y R O G Y R W G B O B Y	NO

노트

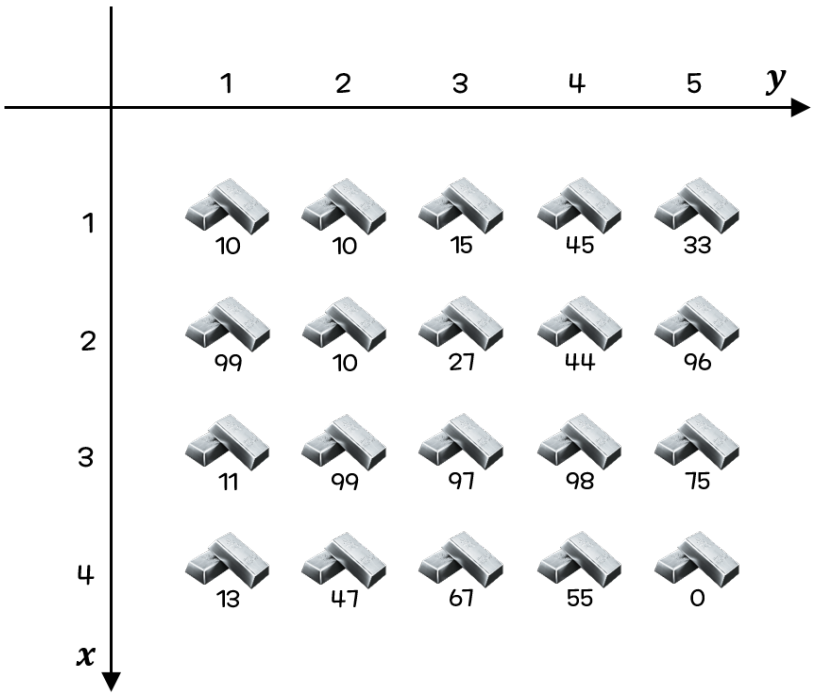
첫 번째 예시의 조각들로는 아래와 같은 색 배치의 큐브를 만들 수 있다.



F. 실버런

실버런(Silver Run)은 2차원 맵에서 캐릭터를 조종해서 최대한 많은 실버를 모으는 것이 목적인 모바일 게임이다.

실버런의 맵에는 실버 주머니들이 상하좌우로 일정한 간격을 두고 N행 M열의 직사각형 모양으로 배치되어 있으며, 각 주머니에는 정해진 개수만큼의 실버가 들어 있다. 편의상 상하좌우로 이웃한 실버 주머니 사이의 간격을 **한 칸**이라고 표현하며, 가장 왼쪽 위 주머니로부터 왼쪽으로 한 칸, 위로 한 칸 떨어진 위치로부터 아래로 x칸, 오른쪽으로 y칸 떨어진 위치를 (x, y)라고 표현한다. 즉 가장 왼쪽 위 주머니의 위치는 $(1, 1)$, 가장 오른쪽 아래 주머니의 위치는 (N, M)이다.



게임이 시작되면 실버 주머니들은 1초당 한 칸의 일정한 속도로 왼쪽으로 움직이며, 유저는 캐릭터를 이동시키며 캐릭터와 부딪히는 실버 주머니에 든 실버를 획득하게 된다. 캐릭터와 주머니의 크기는 충분히 작아서, 실버를 획득하기 위해서는 캐릭터와 주머니의 위치가 정확히 일치해야 한다.

이 게임에서 가장 좋은 캐릭터는 16silver라는 캐릭터이다. 16silver는 임의의 위치에서 출발할 수 있으며(정수 좌표가 아니어도 된다!), 1초당 한 칸의 일정한 속도로 위, 아래 또는 오른쪽으로 이동할 수 있다. 그러나 제자리에 멈춰 있을 수는 없으며, 1초 단위로만 이동 명령을 내릴 수 있다. 예를 들어 "위로 2.5초 동안 이동"과 같은 명령은 내릴 수 없다.

맵이 주어질 때, 16silver 캐릭터를 이용해 모을 수 있는 실버의 최대 수량을 구하는 프로그램을 작성해 보자.

입력

첫 번째 줄에 맵에 있는 실버 주머니들이 이루는 직사각형의 행과 열 수를 나타내는 정수 N, M($1 \le N$, $M \le 1,000$)이 공백을 사이에 두고 주어진다.

다음 N개의 줄에 걸쳐 각 줄에 M개의 정수가 주어진다. i번째 줄의 j번째 정수는 (i, j) 위치에 있는 실버 주머니에 들어 있는 실버의 양이며, 입력되는 모든 정수는 0 이상 $100,000$ 이하이다.

출력

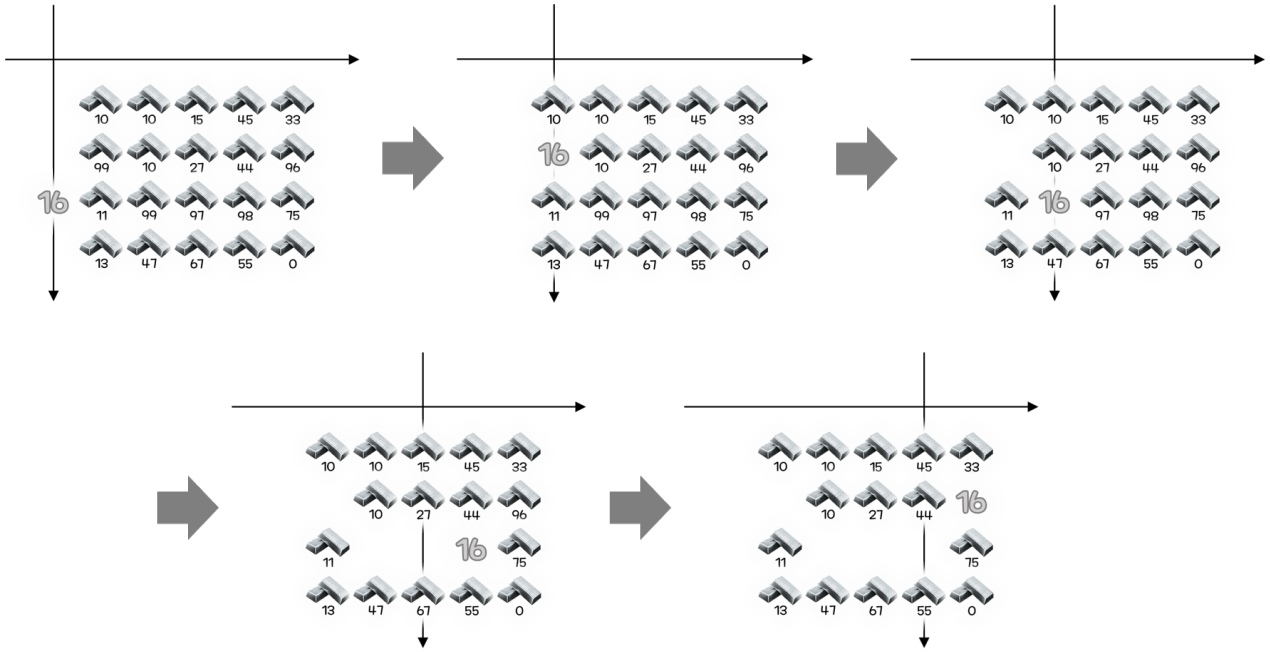
첫 번째 줄에 16silver 캐릭터로 모을 수 있는 실버의 최대 수량을 출력한다.

예제 입출력

standard input	standard output
4 5 10 10 15 45 33 99 10 27 44 96 11 99 97 98 75 13 47 67 55 0	489
5 5 10 14 13 12 11 10 22 16 12 83 95 44 56 98 71 10 97 96 25 99 10 10 47 38 12	469

노트

첫 번째 예제의 경우, 아래 그림과 같이 $(3, 0)$ 위치에서 시작하여 위–아래–오른쪽–위 순서대로 1초씩 움직이면 $99 + 99 + 97 + 98 + 96 = 489$ 개의 실버를 얻을 수 있다.



G. PPAP

bryan은 PPAP를 좋아한다. bryan은 어떻게 하면 사람들에게 PPAP를 전파할 수 있을까 고민하던 중 **PPAP 문자열**이라는 것을 고안하게 되었다.

PPAP 문자열은 문자열 `P`에서 시작하여 문자열 내의 `P`를 `PPAP`로 바꾸는 과정을 반복하여 만들 수 있는 문자열로 정의된다. 예를 들어 `PPAP`는 PPAP 문자열이다. 또한 `PPAP`의 두 번째 `P`를 `PPAP`로 바꾼 `PPPAPAP` 역시 PPAP 문자열이다.

문자열이 주어졌을 때, 이 문자열이 PPAP 문자열인지 아닌지를 알려 주는 프로그램을 작성하여라.

입력

첫 번째 줄에 문자열이 주어진다. 문자열은 대문자 알파벳 `P`와 `A`로만 이루어져 있으며, 문자열의 길이는 1 이상 $1,000,000$ 이하이다.

출력

첫 번째 줄에 주어진 문자열이 PPAP 문자열이면 `PPAP`를, 아닌 경우 `NP`를 출력한다.

예제 입출력

standard input	standard output
PPPAPAP	PPAP
PPAPAPP	NP

H. 달빛 여우

관악산 기슭에는 보름달을 기다리는 달빛 여우가 한 마리 살고 있다. 달빛 여우가 보름달의 달빛을 받으면 아름다운 구미호로 변신할 수 있다. 하지만 보름달을 기다리는 건 달빛 여우뿐만이 아니다. 달빛을 받아서 멋진 늑대인간이 되고 싶어하는 달빛 늑대도 한 마리 살고 있다.

관악산에는 1번부터 N번까지의 번호가 붙은 N개의 나무 그루터기가 있고, 그루터기들 사이에는 M개의 오솔길이 나 있다. 오솔길은 어떤 방향으로든 지나갈 수 있으며, 어떤 두 그루터기 사이에 두 개 이상의 오솔길이 나 있는 경우는 없다. 달빛 여우와 달빛 늑대는 1번 나무 그루터기에서 살고 있다.

보름달이 뜨면 나무 그루터기들 중 하나가 달빛을 받아 밝게 빛나게 된다. 그러면 달빛 여우와 달빛 늑대는 먼저 달빛을 독차지하기 위해 최대한 빨리 오솔길을 따라서 그 그루터기로 달려가야 한다. 이 때 달빛 여우는 늘 일정한 속도로 달려가는 반면, 지구력이 부족한 달빛 늑대는 오솔길 하나를 달빛 여우의 두 배의 속도로 달려간 뒤 다음 오솔길은 달빛 여우의 절반의 속도로 걸어가는 것을 반복한다. 달빛 여우와 달빛 늑대 모두 자신의 이동 방법을 잘 알고 있으며 각자 시간이 가장 적게 걸리는 경로로 이동한다. 따라서 둘의 이동 경로가 서로 다를 수도 있다.

출제자는 관악산의 모든 동물을 사랑하지만, 이번에는 달빛 여우를 조금 더 사랑해 주기로 했다. 그래서 달빛 여우가 달빛 늑대보다 먼저 도착할 수 있는 그루터기에 달빛을 비춰 주려고 한다. 이런 그루터기가 몇 개나 있는지 알아보자.

입력

첫 줄에 나무 그루터기의 개수와 오솔길의 개수를 의미하는 정수 N, $M(2 \le N \le 4,000, 1 \le M \le 100,000)$이 주어진다.

두 번째 줄부터 M개의 줄에 걸쳐 각 줄에 세 개의 정수 a, b, $d(1 \le a, b \le N, a \ne b, 1 \le d \le 100,000)$가 주어진다. 이는 a번 그루터기와 b번 그루터기 사이에 길이가 d인 오솔길이 나 있음을 의미한다.

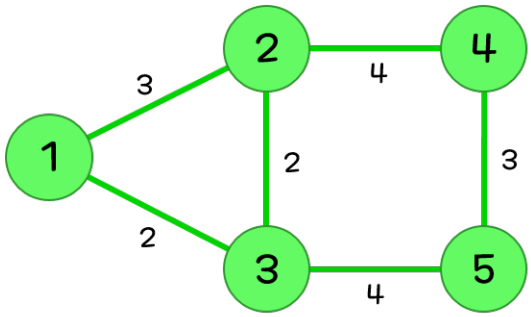
출력

첫 줄에 달빛 여우가 달빛 늑대보다 먼저 도착할 수 있는 나무 그루터기의 개수를 출력한다.

예제 입출력

standard input	standard output
5 6 1 2 3 1 3 2 2 3 2 2 4 4 3 5 4 4 5 3	1

노트



5번 그루터기에 달빛을 비추면 달빛 여우가 달빛 늑대보다 먼저 도착할 수 있다. 4번 그루터기에 달빛을 비추면 달빛 여우와 달빛 늑대가 동시에 도착한다.

I. 뚜루루 뚜루

최근 **아기 석환**이라는 노래가 큰 인기를 끌고 있다. 이 노래는 귀여운 아기 석환 캐릭터가 등장하는 동영상과 중독성 있는 **뚜루루 뚜루** 후렴구로 전국 각지의 학생들과 직장인들의 마음을 사로잡았다.

강남의 직장인 gs12117 역시 이 노래에 흠뻑 빠졌다. 특히 이 노래의 후렴구에 중독된 gs12117은 R줄 C칸으로 나뉘어진 종이에 **뚜루루 뚜루** 후렴구를 계속해서 적어 나가기 시작했다. 후렴구를 적을 때는 종이의 첫 줄 가장 왼쪽 칸에서부터 오른쪽으로 한 칸에 한 글자씩 **뚜**, **루**, **루**, **뚜**, **루**를 순서대로 적어 나가며, 한 줄의 가장 오른쪽 칸에 도달하면 다음 줄의 가장 왼쪽 칸으로 넘어간다.

뚜	루	루	뚜	루	뚜	루
루	뚜	루	뚜	루	루	뚜
루	뚜	루	루	뚜	루	뚜
루	루	뚜	루	뚜	루	루
뚜	루	뚜	루	루	뚜	루
뚜	루	루	뚜	루	뚜	루
루	뚜	루	뚜	루	루	뚜

7×7 종이에서의 예시

gs12117은 이 종이에서 **뚜루루 뚜루** 경로를 많이 찾으려고 한다. **뚜루루 뚜루** 경로란 임의의 칸에서 출발해서 이미 방문한 칸을 다시 방문하지 않도록 상하좌우로 이동하면서 각 칸에 적힌 글자를 순서대로 읽었을 때, 그 결과가 정확히 **뚜루루 뚜루**가 되는 경로를 말한다.

종이의 크기가 주어졌을 때, gs12117이 찾을 수 있는 **뚜루루 뚜루** 경로의 개수를 구하여라. 어떤 두 경로가 같은 순서에 다른 칸을 방문할 경우 두 경로는 서로 다른 경로이다.

입력

첫 줄에 종이의 줄 수와 칸 수를 의미하는 정수 R과 $C(1 \le R, C \le 12,117)$가 주어진다.

출력

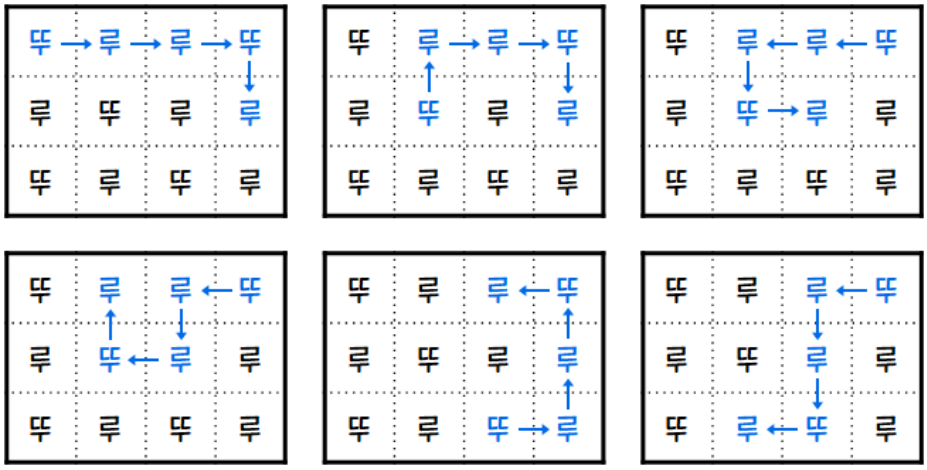
첫 줄에 gs12117이 찾을 수 있는 **뚜루루 뚜루** 경로의 개수를 출력한다.

예제 입출력

standard input	standard output
3 4	23
5 7	162

노트

첫 번째 예시에서는 다음과 같은 경로들이 가능하다.



J. Cherrypick

SNUPC(SNU Patisserie Cafe)에서는 유명한 케이크를 팔고 있다. 이 케이크는 N개의 행과 N개의 열로 나눌 수 있는 정사각형 모양이다. 그리고 이 케이크를 1×1 크기의 정사각형 조각 N^2개로 잘랐을 때, 각 조각 위에 체리가 하나씩 올려져 있다. i행 j열 조각 위의 체리는 $C_{i,j}$만큼의 당도를 가지고 있다. 이 케이크는 한 번에 먹기에는 너무 크기 때문에, 각 변이 축에 평행하고 각 꼭지점의 x좌표와 y좌표가 정수인 정사각형 모양으로 잘라내어 판매하고 있는데, 잘라낸 정사각형의 한 변의 길이를 X라고 했을 때, 잘라낸 조각을 X^2원에 판매한다.

이 가게에 슈퍼 부자 Corea가 방문하였다. Corea는 이전 가게에서 이미 너무 많은 음식을 먹었기 때문에, 케이크를 산 다음 그 위에 있는 체리 중 가장 단 체리 한 조각만을 먹으려고 한다. 또, 이 케이크의 특정한 조각을 하나 골라서, 그 조각은 반드시 포함되도록 구매하려고 한다.

Corea는 돈을 낭비하는 것을 별로 좋아하지 않기 때문에, 자신이 먹게 될 체리의 당도에서 케이크를 사는데 내는 비용을 뺀 값을 최대화하려고 한다. 하지만 Corea는 그런 사소한 결정을 하는데 낭비할 시간이 없기 때문에, Corea가 케이크에 포함될 조각을 고를 동안 당신은 Corea가 선택할 수 있는 모든 경우의 수에 대해서 Corea가 먹게 될 체리의 당도에서 케이크의 비용을 뺀 값의 최댓값을 계산해야 한다.

입력

첫 번째 줄에 전체 케이크의 행과 열의 길이 $N(1 \le N \le 1,000)$이 주어진다.

두 번째 줄부터 총 N개의 줄에 각각 N개의 정수 $C_{i,j}$ $(1 \le C_{i,j} \le 1000)$가 주어지는데, 이는 i행 j열 조각 위에 놓인 체리의 당도이다.

출력

공백으로 구분된 N개의 정수를 N줄에 걸쳐 출력한다. i번째 줄의 j번째 정수로 출력하는 값은 i행 j열 조각을 포함하도록 케이크를 구매했을 때 Corea가 먹게 될 체리의 당도에서 케이크의 비용을 뺀 값의 최대값이다.

예제 입출력

standard input	standard output
2 2 1 4 1	1 0 3 0
3 1 2 3 4 5 6 7 8 9	1 2 2 4 5 5 6 7 8