Скрудж Макдък е загрижен, не за живота си, а за това дали събраните от него монети ще бъдат достатъчни. Вашата задача е да му кажете колко монети ще събере, следвайки правилата по-горе.

**Входни данни**

* Четете от конзолата.
* На първия ред ще получите N и M (размера на лабиринта)
* На следващите N реда ще получите M на брой числа (integer)
* Входните данни винаги ще бъдат валидни и няма нужда да се проверяват експлицитно
* Началната позиция на Скрудж Макдък ще бъде маркирана като единственото число 0

**Изходни данни**

* Отпечатайте на конзолата.
* На един ред изведете броя на монетите, които Скрудж Макдък може да събере, спазвайки правилата.

**Ограничения:**

* 2 <= N <= 10
* 2 <= M <= 10
* Всяка клетка може да съдържа до 1024 монети

**Примерни тестове**

**Входни данни**

4 3

3 2 4

2 0 3

1 1 5

2 2 5

**Изходни данни**

22

**Входни данни**

3 3

10 10 0

10 10 10

10 10 10

**Изходни данни**

78

**Входни данни**

3 3

10 10 10

10 0 10

10 10 10

**Изходни данни**

80

**Входни данни**

2 3

0 5 2

2 5 3

**Изходни данни**

15

import java.util.Scanner;

public class Scrooge {

private static int[][] labyrinth;

private static int rows;

private static int columns;

private static int coinsCollected;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

rows = scanner.nextInt();

columns = scanner.nextInt();

labyrinth = new int[rows][columns];

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

labyrinth[i][j] = scanner.nextInt();

}

}

findExit();

System.out.println(coinsCollected);

}

private static void findExit() {

int currentRow = 0;

int currentColumn = 0;

while (true) {

int nextRow = -1;

int nextColumn = -1;

int maxCoins = 0;

// Check for possible moves.

for (int i = -1; i <= 1; i++) {

for (int j = -1; j <= 1; j++) {

int nextRowCandidate = currentRow + i;

int nextColumnCandidate = currentColumn + j;

if (isValidMove(nextRowCandidate, nextColumnCandidate)) {

int coinsInCell = labyrinth[nextRowCandidate][nextColumnCandidate];

if (coinsInCell > maxCoins) {

maxCoins = coinsInCell;

nextRow = nextRowCandidate;

nextColumn = nextColumnCandidate;

} else if (coinsInCell == maxCoins) {

// Prefer left, right, up, down order.

if (nextRowCandidate == currentRow && j == 1) {

nextRow = nextRowCandidate;

nextColumn = nextColumnCandidate;

} else if (nextColumnCandidate == currentColumn && i == 1) {

nextRow = nextRowCandidate;

nextColumn = nextColumnCandidate;

} else if (nextRowCandidate == currentRow && j == -1) {

nextRow = nextRowCandidate;

nextColumn = nextColumnCandidate;

} else if (nextColumnCandidate == currentColumn && i == -1) {

nextRow = nextRowCandidate;

nextColumn = nextColumnCandidate;

}

}

}

}

}

// Collect coins and move to the next cell.

coinsCollected += labyrinth[nextRow][nextColumn];

labyrinth[nextRow][nextColumn] = 0;

currentRow = nextRow;

currentColumn = nextColumn;

// Check if we are surrounded by empty cells.

if (isSurroundedByEmptyCells(currentRow, currentColumn)) {

break;

}

}

}

private static boolean isValidMove(int row, int column) {

return row >= 0 && row < rows && column >= 0 && column < columns;

}

private static boolean isSurroundedByEmptyCells(int row, int column) {

for (int i = -1; i <= 1; i++) {

for (int j = -1; j <= 1; j++) {

int nextRow = row + i;

int nextColumn = column + j;

if (isValidMove(nextRow, nextColumn) && labyrinth[nextRow][nextColumn] > 0) {

return false;

}

}

}

return true;

}

}