F. Велодорожки

|  |  |
| --- | --- |
| Ограничение времени | 4 секунды |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Мэр одного города очень любит следить за тенденциями и воспроизводить их в своём городе. До него дошла новость о популярности велодорожек. Теперь он хочет проложить велодорожки в своём городе и сделать это лучше, чем в других городах! Поэтому он решил сделать велодорожки даже на главной площади города.

Главная площадь представляет собой прямоугольник шириной w и высотой h, замощённый квадратными плитками со стороной 1. Мэр хочет, чтобы было проложено две велодорожки **одинаковой ширины**: одна горизонтальная и одна вертикальная. К сожалению, ремонт на площади проводился достаточно давно и на некоторых плитках уже появились трещины. Мэр хочет проложить велодорожки так, чтобы после этого на площади остались только целые плитки. При строительстве велодорожек плитки на их месте убираются. Можно только убирать плитки с площади и нельзя менять местами или добавлять новые. Чтобы потратить меньше денег, мэр хочет сделать велодорожки наименьшей возможной ширины, при этом ширина дорожек должна быть целым числом. Определите, какой должна быть ширина велодорожек.

Формат ввода

В первой строке входных данных содержатся три целых числа w,h,n (1≤w,h≤109, 1≤n≤min(w×h,3⋅105))  — ширина и высота площади и количество потрескавшихся плиток соответственно.

В следующих n строках содержится по 2 целых числа xi,yi (1≤xi≤w, 1≤yi≤h)  — координаты потрескавшихся плиток. (xi,yi)≠(xj,yj) при i≠j.

Формат вывода

Выведите единственное число c (1≤c≤min(w,h))  — наименьшую возможную ширину велодорожек.

Пример 1

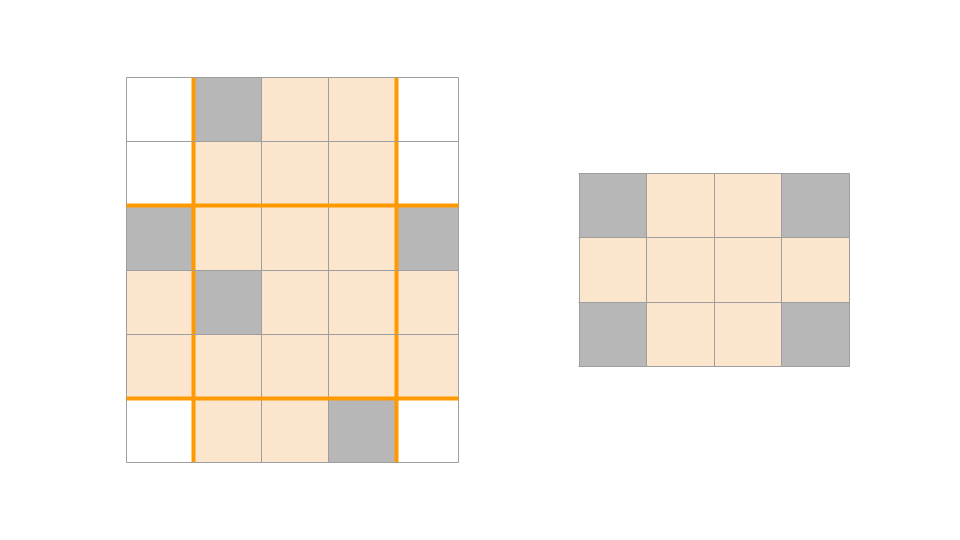
| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5 6 5  5 4  2 6  4 1  2 3  1 4 | 3 |

Пример 2

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 4 3 4  1 1  4 3  4 1  1 3 | 3 |

Примечания

Ниже приведены картинки к примерам из условия. Серым отмечены потрескавшиеся плитки. Во втором примере ширина дорожек равна меньшей из сторон прямоугольника.



using System;

using System.IO;

public struct Point

{

public int X;

public int Y;

public Point(int x, int y)

{

X = x;

Y = y;

}

public override bool Equals(object? obj)

{

return obj is Point ? (Point)obj == this: false;

}

public static bool operator == (Point p1, Point p2)

{

return p1.X == p2.X && p1.Y == p2.Y;

}

public static bool operator != (Point p1, Point p2)

{

return p1.X != p2.X || p1.Y != p2.Y;

}

public override int GetHashCode()

{

return X.GetHashCode() \* Y.GetHashCode();

}

}

internal class Program

{

public static void Main()

{

StreamReader reader = new StreamReader("input.txt");

string[] input1 = reader.ReadLine().Split();

int n = Convert.ToInt32(input1[0]);

int m = Convert.ToInt32(input1[1]);

int N = Convert.ToInt32(input1[2]);

if (N == 1 || N == 2)

{

Console.WriteLine(1);

return;

}

Point[] points = new Point[N];

for (int i = 0; i < N; i++)

{

string[] input2 = reader.ReadLine().Split();

points[i] = new Point(Convert.ToInt32(input2[0]) - 1, Convert.ToInt32(input2[1]) - 1);

}

int left = 1;

int right = Math.Min(n, m);

while (left < right)

{

int mid = (left + right) / 2;

int[] koordForRow1 = new int[2];

int[] koordForColumn1 = new int[2];

int[] koordForRow2 = new int[2];

int[] koordForColumn2 = new int[2];

bool find = false;

int firstRow1 = -1;

int firstColumn1 = -1;

int firstRow2 = -1;

int firstColumn2 = -1;

bool firstFlag = true, secondFlag = true;

for (int i = 0; i < N; i++) // pаботаем с индексами

{

if (firstRow1 == -1)

{

firstRow1 = points[i].X;

if (firstRow1 + mid - 1 >= n)

{

koordForRow1[1] = n - 1 - firstRow1;

koordForRow1[0] = firstRow1 - (n - mid);

firstRow1 = n - mid;

}

else

{

koordForRow1[1] = mid - 1;

koordForRow1[0] = 0;

}

firstColumn2 = points[i].Y;

if (firstColumn2 + mid - 1 >= m)

{

koordForColumn2[1] = m - 1 - firstColumn2;

koordForColumn2[0] = firstColumn2 - (m - mid);

firstColumn2 = m - mid;

}

else

{

koordForColumn2[1] = mid - 1;

koordForColumn2[0] = 0;

}

}

else

{

if (firstFlag)

{

if (points[i].X >= firstRow1 && points[i].X <= firstRow1 + mid - 1)

{

koordForRow1[0] = Math.Min(points[i].X - firstRow1, koordForRow1[0]);

koordForRow1[1] = Math.Min(firstRow1 + mid - 1 - points[i].X, koordForRow1[1]);

firstRow1 = Math.Min(firstRow1, points[i].X);

}

else if (firstRow1 - points[i].X > 0 && firstRow1 - points[i].X <= koordForRow1[1])

{

koordForRow1[0] = points[i].X;

koordForRow1[1] = koordForRow1[1] - firstRow1 - points[i].X;

firstRow1 = Math.Min(points[i].X, firstRow1);

}

else

{

if (firstColumn1 == -1)

{

firstColumn1 = points[i].Y;

if (firstColumn1 + mid - 1 >= m)

{

koordForColumn1[1] = m - 1 - firstColumn1;

koordForColumn1[0] = firstColumn1 - (m - mid);

firstColumn1 = m - mid;

}

else

{

koordForColumn1[1] = mid - 1;

koordForColumn1[0] = 0;

}

}

else

{

if (points[i].Y >= firstColumn1 && points[i].Y <= firstColumn1 + mid - 1)

{

koordForColumn1[0] = Math.Min(points[i].Y - firstColumn1, koordForColumn1[0]);

koordForColumn1[1] = Math.Min(firstColumn1 + mid - 1 - points[i].Y, koordForColumn1[1]);

firstColumn1 = Math.Min(firstColumn1, points[i].Y);

}

else

{

if (firstColumn1 - points[i].Y > 0 && firstColumn1 - points[i].Y <= koordForColumn1[1])

{

koordForColumn1[0] = points[i].Y;

koordForColumn1[1] = koordForColumn1[1] - firstColumn1 - points[i].Y;

firstColumn1 = points[i].Y;

}

else firstFlag = false;

}

}

}

}

if (secondFlag)

{

if (points[i].Y >= firstColumn2 && points[i].Y <= firstColumn2 + mid - 1)

{

koordForColumn2[0] = Math.Min(points[i].Y - firstColumn2, koordForColumn2[0]);

koordForColumn2[1] = Math.Min(firstColumn2 + mid - 1 - points[i].Y, koordForColumn2[1]);

firstColumn2 = Math.Min(firstColumn2, points[i].Y);

}

else if (firstColumn2 - points[i].Y > 0 && firstColumn2 - points[i].Y <= koordForColumn2[1])

{

koordForColumn2[0] = points[i].Y;

koordForColumn2[1] = koordForColumn2[1] - firstColumn2 - points[i].Y;

firstColumn2 = Math.Min(points[i].Y, firstColumn2);

}

else

{

if (firstRow2 == -1)

{

firstRow2 = points[i].X;

if (firstRow2 + mid - 1 >= n)

{

koordForRow2[1] = n - 1 - firstRow2;

koordForRow2[0] = firstRow2 - (n - mid);

firstRow2 = n - mid;

}

else

{

koordForRow2[1] = mid - 1;

koordForRow2[0] = 0;

}

}

else

{

if (points[i].X >= firstRow2 && points[i].X <= firstRow2 + mid - 1)

{

koordForRow2[0] = Math.Min(points[i].X - firstRow2, koordForRow2[0]);

koordForRow2[1] = Math.Min(firstRow2 + mid - 1 - points[i].X, koordForRow2[1]);

firstRow2 = Math.Min(firstRow2, points[i].X);

}

else

{

if (firstRow2 - points[i].X > 0 && firstRow2 - points[i].X <= koordForRow2[1])

{

koordForRow2[0] = points[i].X;

koordForRow2[1] = koordForRow2[1] - firstRow2 - points[i].X;

firstRow2 = points[i].X;

}

else secondFlag = false;

}

}

}

}

}

}

if (firstFlag || secondFlag)

{

right = mid;

}

else

{

left = mid + 1;

}

}

Console.WriteLine(left);

}

}