Rod Binding

Objective
Give students practice with Minimum Spanning Trees

Story
Upon casting the final incantation the floor begins to erode away and a uniform spaced grid of floor divots begin to materialize. The ground shakes and poles rise from the ground from some of the divots. A gold chain drops from the ceiling into the center of the room. “Great another puzzle.”

After some experimenting you find the by connecting the poles with the gold chain a hidden door opens. You want to use as little chain as possible (in case you need some for later; you’re totally not going to pawn it when you get back to town), and when one of the chain segments is removed the door closes. You tried to connect three poles together by connecting a golden chain to another, but that didn’t work. You also tried to cross two of golden chains such that they touch in the center, but that also did not keep the hidden door open. You had to have exactly two poles connected by each chain segment.

Normally the task of determining the least amount of chain would be fairly straight forward, but one of the poles is loose, and you notice that you can move it to a different divot with the door remaining open. The only caveat is that the pole must be placed in a divot. You want to know what is the least amount of gold chain required to keep the hidden door open given that you have one pole that can be placed at any location.

Problem
Given the dimensions of the grid, and the locations of the poles that cannot move determine the least amount of gold chain that is required to connect them AND the one loose pole.

Input
Input will being with 3 integers, n, c, and r ($1 \le n \le 9{,}999$; $1 \le r, c \le 100$; $n < r \times c$), representing the number of poles stuck in a divot, the number of rows, and the number of columns in the grid of divots. The following n lines will each contain two integers, the i-th of which contains x_i and y_i ($1 \le x_i \le c$; $1 \le y_i \le r$), representing the column and the row of i-th pole respectively. The loose pole can only be placed in a location with a divot, and there are no divots outside the grid.

Output
Output the sum of the gold chain required to connect all the poles including the one pole not listed in the input. Consider the distance between divots adjacent in the 4 cardinal directions to be 1 unit long. An answer will be correct if it has a relative margin of error less than 10^{-6}.

Sample Input	Sample Output
4 10 10 1 1 5 5 5 1 1 5	11.313708498984761
3 5 1 1 3 1 4 1 2	3.0

Explanation

Case 1

The optimal location for the loose rod is position (3, 3). This allows us to connect all the other rods to the loose rod. That means the total answer would be
distance((1,1), (3,3)) + distance((5,5), (3,3)) + distance((5,1), (3,3)) + distance((1,5), (3,3)) =
$\sqrt{8} + \sqrt{8} + \sqrt{8} + \sqrt{8} \approx 11.313708498984761$
Valid answers would include values from
11.313708498984761 X $(1-1^{-6})$ = 11.3136971853
to
11.313708498984761 X $(1+1^{-6})$ = 11.3137198127

Case 2

The optimal location for the loose rod is any open position (i.e. (1, 5) or (1, 1). They both will result in the same answer.

If (1, 1) is selected the amount of gold used would be:
1 (distance between (1, 1) and (1, 2)) +
1 (distance between (1, 2) and (1, 3)) +
1 (distance between (1, 3) and (1, 4))

If (1, 5) is selected the amount of gold used would be:
1 (distance between (1, 2) and (1, 3)) +
1 (distance between (1, 3) and (1, 4)) +
1 (distance between (1, 4) and (1, 5))

Hints

Distance Formula: Remember the distance formula is the square root of the sum of the squares of the dx and dy.

Graph Representation: Let's ignore the unplaced rod for now. We can turn the rod layout into a graph, where each node is a rod, and the edge weights are the distance between the rods. We are trying to minimize the sum of the gold between selected edges. Selecting an edge will always reduce the gold, so we should select edges that minimally connect all the nodes. This is an MST!

Adding back the unplaced rod: We could try all possible placements of the unplaced rod. When placing the rod we know its location and can run MST. From all those possible placements we choose the one that makes the Minimum MST.

THE WORST CASE SCENARIO: In the worst possible outcome we could have 5000 places to place the rod (i.e. how many MSTs we create), and 5000 rods placed. The number of edges in the graph would be 5001 choose 2. Each MST would take about log(5000) * 5000 * 5000 / 2. The total runtime would be over 5000^3 operations. This means that the runtime would be over 1200 seconds. That will not be fast enough for the largest cases trust me.

How to get better than $O(N^3)$: As it turns out the MST without the loose rod is very close to the correct MST. The only possible edges you would consider adding are the edges between the loose pole and the non-loose rod. There is **never** a reason to add edges outside of the original MST that are between "fixed" rods. This means that the edge set would be equal to MST + edges to the loose rod. The total number of edges is 2N. Thus the total runtime for evaluating a possible rod placement is $O(\log(N) * N^2)$ by using the speed up of limiting edges.

Grading Criteria

- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of rods:”)
    - 10 points
- Creates edges by using the distance between rods
    - 10 points
- Sorts the edges and greedily takes the smallest (the way MST algorithms work)
    - 10 points
- Uses some “method” to try all valid possible placements for the loose rod
    - 5 points
- Programs will be tested on 10 cases
    - 5 points each

No points will be awarded to programs that do not compile using javac.

Sometime a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use a Minimum Spanning Tree finding algorithm such as Kruskals or Prims. ***Without this programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.