The Road not Bombed

“Two roads diverged in a yellow wood,
And sorry I could not travel both
Because one was carpet bombed”
— Robert Frost, probably

1. Statement

You’ve recently enacted a new plan to boost your country’s economy: build roads! To facilitate trades, you want to build state-of-the-art roadways between pairs of cities that you have determined are the most important to connect. Two adjacent cities that both have roadways allow civilians to travel quickly between them. You’ve drawn up a road plan to connect all of the cities you want to connect, allowing for travel from a city to its partner using only your new roadways.

However, your spy reveals that an ambitious enemy country wants to bomb your city soon! They’ve pinpointed certain cities they think are crucial and have already begun preparations to bomb these locations. At first you don’t think much of this threat, until you realize it might disrupt your trade logistics!

If a city is bombed, it will undoubtedly disable any roadways you build there. You decide it is of utmost importance that roads still grant access between your designated cities, even if some roads are destroyed by bombs. You could do this by building more roads and creating redundant paths, so even if some roads and paths are destroyed, at least one path will probably stay intact: you only need one valid path of roads to connect a pair of cities. But this requires building more roads than necessary: that’s terrible for the economy!

Thankfully, your spy might be able to give you some information about what coordinates will be bombed. If you give them a set of locations where you plan to build roads, they’ll be able to discreetly ask the enemy about the effect the bombs would have on your connections.

Specifically, they’ll be able to find out if your road plan will be able to withstand the bombing attack and keep all connections intact, or if at least one connection will be broken. However, your spy can only do this so many times, so you’ll probably be working with incomplete information, and not know exactly what is the optimal path to build. But fear not; you figure that with this information, you can still try to cut down on the number of roads you use as much as possible.

2. Formal Description

The grid is of size 16 × 16 (number of rows and columns respectively). Each coordinate represents a city. All coordinates are 0-indexed, so the top left corner is position (0, 0) and the bottom right is (15, 15). You are given p pairs of coordinates, where each pair represents two locations that you want to connect with roads. All such locations are located on the edge of the grid. That is, their coordinates have 0 or 15 in at least one of the x or y coordinate. All coordinates given in pairs are distinct. Finally, each coordinate of the grid has an independent probability of bd (bomb density) of containing a bomb.

A query consists of a list of coordinates, representing all of the roads in a road plan. When a query is passed from the program to the autograder, the autograder first removes all duplicate coordinates, as well as any coordinates that are in the list b, to determine a list of non-bombed roads. Then, the autograder performs a graph search to determine if all p pairs of cities are connected. A pair of cities is connected if there exists a sequence of orthogonally adjacent cells of non-bombed roads between the two coordinates in a path pair. Note that a coordinate can be part of more than one path. If all paths are connected, the query's report is true; if a single path is nonconnected, it is false.

Your goal is to find a road plan with the minimum number of roads such that the query's report returns true. It is guaranteed that some valid road plan exists. Your program is allowed 100 queries, for you are to use previous query results to improve future query results. Ultimately, your raw score will be the minimum size of the road plan over successful queries you make.

3. Code

3.1 Parameters

Grid size: 16 x 16
Path pairs: p = 1 or p = 5
Number of queries: q = 100
Bomb density: bd = 0.1 or 0.25

3.2 Implementation

You are to implement a class with two functions, `setup` and `query`.

`setup(pairs, bd)` takes in a list `pairs` and a float `bd`. The float `bd` is the density of the bombs generated. The list `pairs` has length p = 1 or p = 5, and each item of `pairs` is of the form [[x1, y1], [x2, y2]] where [x1, y1] is the coordinate of one city and [x2, y2] is the coordinate of another city, and these two cities are to be connected by a row. The return value of setup is irrelevant.

`query(q, queryOutputs)` takes in an int q and a list queryOutputs. The int `q` is the number of queries remaining, which is between 1 and 100 inclusive.

`queryOutputs` is an array of length 100 - q containing booleans, which represent the autograder's response to each previous query in order, with the response to the most recent query at the end of the list.

`query` should return a road plan represented by a list of coordinates, where each coordinate represents a built road. The list may be in any order and duplicates will be ignored. Each coordinate should be represented with a length 2 list. Therefore, the road plan will look something like [[x1, y1], [x2, y2], …, [xm, ym]].

3.2 Execution

The autograder begins with parameters p and bd, and randomly generates a 16 x 16 grid of bombs and pairs, while ensuring that a valid road plan exists. It calls `setup(pairs, bd)` to set up your program with the appropriate parameters.

Then, for 100 times, it calls `query(q, queryOutputs)`. Note that on the first call, q = 100, and q decreases to 1. Your program thus makes 100 queries for road plans. Query outputs for all previous queries are contained in `queryOutputs`.

After 100 queries have been made, your raw score is the minimum number of roads used in any valid road plan submitted as a query. If you submitted no valid road plans, your score is 256 (equivalent to a road plan that builds a road at every position).

3.3 Timeout and memory

Your program will have 5 seconds to complete all 100 queries. Your program may use up to 256 MB of memory. If your program uses up all the time, no more queries will be made and the score will be the best one from all previous queries.

3.4 Grading

There are four different configurations of the problem, corresponding to p = 1 or p = 5 and bd = 0.1 or bd = 0.25. Each configuration is graded separately.

On a single configuration, your code will be run some N number of times, where N depends on the number of test cases available. For local testing you can set N to your liking. For the live leaderboard you will be tested on N = 20 test cases, and your raw score will be the geometric mean of the 20 test cases. For the final leaderboard N = 50 test cases will be used. Your final score is obtained by normalizing your raw score in the following manner: your final score is the best raw score (over all contestants) divided by your raw score.

The final scores from the four configurations will be summed to produce a single final score for the entire problem.