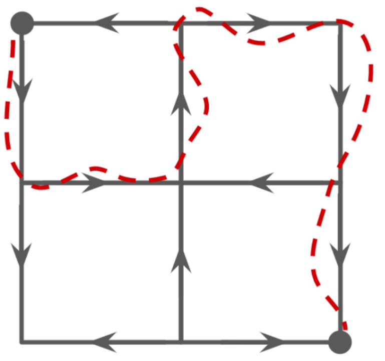
# Riddler April 9, 2021: One-way streets

Mark Girard

April 11,2021

**Question 1.** *In Riddler City, all the streets are currently two-way streets. But in an effort to make the metropolis friendlier for pedestrians and cyclists, the mayor has decreed that all streets should be one-way. Meanwhile, the civil engineer overseeing this transition is not particularly invested in the project and will be randomly assigning every block of each street a random direction.*

*For your daily commute to work, you drive a car two blocks east and two blocks south, as shown in the diagram below. What is the probability that, after each block is randomly assigned a one-way direction, there will still be a way for you to commute to work while staying within this two-by-two block region (i.e., sticking to the 12 streets you see in the diagram)? Here is one such arrangement of one-way streets that lets you commute to work:*



I was not able to think of a solution that did not involve brute force to generate all possible traffic patterns of the $2 \times 2$ traffic grid. As there are a total of twelve segments of road and each road can have one of two directionalities, there are a total of $2^{12}$ (4096) possibilities.

My method was to generate each possibility as a directed graph then for each of the 4096 directed graphs determine if there is a path on the graph from the upper-left corner to the lower-right corner. Once a directed graph has been constructed, I used a breadth-first algorithm to find all nodes that can be visited from the initial node. The graph in question has 9 nodes (it is a $3 \times 3$ grid of intersections), but the following code works for any $m \times n$ rectangular grid of intersections.

The code below defines the following functions:

- `makeEdges(m,n)` — Makes a list having length $2mn - m - n$ of all one-way edges in the grid. Each edge either has the form $[(i, j), (i + 1, j)]$ or $[(i, j), (i, j + 1)]$, where $[(a, b), (c, d)]$ indicates an edge connecting vertices at $(a, b)$ and $(c, d)$.

- `makeNeighbours(edges, directions)` — Given a list of edges from `makeEdges` and a string of zeros and ones having the same length, this function loops through all edgess and flips edge `k` if `direction[k]` is equal to 1.

- `existsPath(start,end,neighbours)` — Determines if there is a path from vertex `start` to vertex `end` in the directed graph whose directed edges are listed in `neighbours`.

- `solution(m,n)` — Computes the number of total possibilites for the graph and the number of possibilities having a route from the upper-left corner to the lower-right corner.

The output if `solution(3,3)` is `1135, 4096`. Thus, given a random choice of directions for each of the one-way streets, there is a 1135/4096 (i.e., 0.277099609375) chance of their existing a route from home to work.

```
from collections import defaultdict

def makeEdges(m,n):
    edges = []
    for i, j in prodict(range(m),range(n)):
        if i < m - 1 :
            edges.append([(i,j),(i+1,j)])
        if j < n - 1 :
            edges.append([(i,j),(i,j+1)])
    return edges

def makeNeighbours(edges, directions):
    neighbours = defaultdict(set)
    for k, [a, b] in enumerate(edges):
        if int(directions[k]):
            a, b = b, a
        neighbours[a].add(b)
    return neighbours

def existsPath(start,end,neighbours):
    visited = set([start])
    nodes = [start]
    while nodes:
        new_nodes = []
        for node in nodes:
            for neighbour in neighbours[node]:
                if neighbour == end:
                    return True
                if (neighbour not in visited):
                    visited.add(neighbour)
                    new_nodes.append(neighbour)
        nodes = new_nodes
    return False
```

```
def solution(m,n):
    edges = makeEdges(m,n)
    feasible = []
    num_edges = len(edges)
    count = 0
    for i in range(2**num_edges):
        neighbours = makeNeighbours(edges,format(i, '0'+str(num_edges)+'b'))
        if existsPath((0,0),(m-1,n-1),neighbours):
           count += 1
           feasible.append(i)
    return count, 2**num_edges

a,b = solution(3,3)
print(a,b,a/b)
```