1. A new graph will be made from the old one that looks the exact same except that the vertices will also hold the amount of steps taken so far mod 3. That way it will be easy to find the shortest path where that value is 0
    - Vertices: {(v, i) : v ∈ V(G) and i = ∈ {0,1,2}}
    - Edges: $(v_1, i_1)$ -> $(v_2,i_2)$ (directed)
    - Values: none or all values are 1
    - Problem: Shortest path
    - Algorithm: BFS
    - Time: Θ(edges + vertices)
2. I'll use indexes to describe where in the maze I'm looking at any given step. i to describe horizontally and j to describe vertically. Using i and j I'll be able to get a value for that square and be able to see where in the maze it could go from there by adding or subtracting from either i or j
    - Vertices: M[i, j] where i and j are indices for the matrix
    - Edges: $(i_1, j_1)$ -> $(i_1, j_2)$ or $(i_1, j_1)$ -> $(i_2, j_1)$
    - Values: amount to add to i or j. gotten from M[i, j]
    - Problem: shortest path
    - Algorithm: BFS
    - Time: Θ(edges + vertices)
3. Road[1..n, 1..n] is a 2d array of chances of being abducted by aliens between every city. Starting in city i and going to city j has Road[i, j] chance of driving there safely. To find the safest route I should find the max( $\Pi_{(i, j) \in P}$ Road[i, j] ) where P is a path from one city to another one. Taking the log of that function gets max( $\Sigma_{(i, j) \in P}$ (log (Road[i, j]) ) ). To find the minimum length change it to a minimum problem. Min( $\Sigma_{(i, j) \in P}$ (-log (Road[i, j]) ) ). Use bellman-ford to find the value then use 2^that value to reverse the process.
    - Vertices: cities
    - Edges: i -> j for all i,j ∈ cities. directed
    - Values: -log( Road[i, j])
    - Problem: shortest path
    - Algorithm: bellman ford
    - Time: O(n^2 + n^3)