We reduce from 3-SAT. Problem is often known as Vertex-Disjoint Paths.

Construction: We are given an input to 3-SAT with m clauses and n variables. Label the *variables* (as opposed to literals) appearing in clause j as x_{1j}, x_{2j}, and x_{3j}. For i from 1 to n, and for j from 1 to m, create two vertices, x_{ij}^T and x_{ij}^F. This will result in six vertices per clause. Next, for each clause j, we will construct a city/safe zone pair, c_j^C and s_j^C (C for "clause"). Finally, for each variable i, we will construct a city/safe zone pair, c_i^V and s_i^V (V for "variable").

Now for each clause, connect c_j^C to x_{ij}^T if the variable x_{ij} appears unnegated in clause j, and connect c_j to x_{ij}^F is x_{ij} appears negated. Then connect whichever variable vertex you used (x_{ij}^T or x_{ij}^F) to s_j^C. In other words, there will be three paths of length 2 from c_j^C to s_j^C for every j. Whichever one of these paths we use will be the literal evaluating to T in clause j.

Next, for each variable x_i, connect c_i^V to $x_{ij^1}^T$, where j^1 is the lowest-indexed clause where variable x_i appears. Then, connect $x_{ij^1}^T$ to the $x_{ij^2}^T$, where j^2 is the second-lowest indexed appearance of x_i, and so on, ending the path constructed this way at s_i^V. Do the same for the x_{ij}^F vertices, connecting the instances of x_i along a path from c_i^V to s_i^V. (In other words, there will be two paths from c_i^V to s_i^V, a "True" path, and a "False" path. Whichever one of these paths we *don't* use will correspond to the truth assignment for x_i - this will leave the vertices in the unused path free for use in clause pair paths.

The algorithm to solve 3-SAT is to construct the above graph, call the black box for Vertex-Disjoint Paths, and return the result.

Correctness: Assume there is a satisfying assignment for the given instance of 3-SAT. For each variable i, if i is true, use the path from c_i^V to s_i^V that goes through F vertices, and otherwise, choose the path that goes through T vertices. Next, for each clause j, there is some literal in clause j which evaluates to T. If that literal is a non-negated variable, go through the T vertex for the corresponding variable in clause j's six-vertex gadget. Otherwise, go through the F vertex. We need to check two things:

- Every city/safe zone pair has a path. This follows from the fact that every clause is satisfied and every variable assigned a truth value.
- The paths are vertex disjoint. This could only happen if a clause path crosses a vertex path. This cannot happen, as doing so would have required a clause to be satisfied by a literal which evaluates to F.

Conversely, assume there is a set of vertex-disjoint paths from each city to its safe zone. By construction, each path for a pair corresponding to a variable i can go through either T or F nodes for that variable. If it goes through T nodes, set x_i to F, otherwise set it to T. We need to check two things:

- Every variable is assigned exactly one value. This is by construction - each city/safe zone pair has exactly one path.
- Each clause j has a literal which evaluates to T. For each clause, there is a path from the city to the safe zone node which goes through a literal vertex for some variable i. It must be

that this literal vertex evaluates to T - if it didn't, then the paths for variable i and clause j would cross.

Runtime: This construction involves 8 vertices for each clause and 2 more for each variable, and approximately the same number of edges. We call the black box once, which is clearly polynomial-time.

Flood Warning is in NP: A polynomial-length certificate is the list of paths (linear in the size of the graph), and a polynomial-time certifier algorithm checks that each path is valid, and that no two paths share a vertex. This can be done in linear time.