Ford-Fulkerson Algorithm
CLRS pages 725-727 were utilized for ideas regarding the Ford Fulkerson algorithm.
A We will be constructing a flow network where we will be determining the maximum flow value of |X|. A source node will be denoted as *s* and sink node will be denoted as *t*. The source node *s* will be connected to all of the X nodes separately where there will be a directed edge with unit capacity present from each edge between *s* and X. Here, X will denote the “populated vertices” and S will denote the “safe vertices”. Assuming that X and S are disjoint, we will still assume that there is some path from X to S using some unique edges. Each node in S will then be connected to sink node *t* via a directed edge and with capacity |X|. We will set all flow values, or edge values equal to 1, which we will use to determine the evacuation routes that end at sink node *t*. Given that we will decide in polynomial time whether a set of evacuation routes exist, we will apply the Ford-Fulkerson algorithm where the time-complexity will be in polynomial time. |X| will be set as our capacity between *s* to X, and X and S will be disjoint. After we have calculated max-flow, we will state that max-flow=|X| only if such evacuation routes exist. Since we know that each vertice X will be receiving capacity from *s*, and because there exists edges between X and S, and since the flow is eventually reaching the sink node t, we are able to make the conclusion that max-flow=|X|. If such evacuation routes exist, then each edge will be given a value of 1, denoting that no such congestion exists between nodes of s to X, X to S, and S to *t*. Here, the values of edges between S and *t* will denote the number of evacuation routes that end at sink node *t*.
B To change iii and state that “the paths do not share any vertices”, and to prove this in polynomial time whether such a set of evacuation routes exist, we will modify the algorithm by replacing all the vertexes (Vertex) in G by splitting them into two vertices (Vertex1, Vertex2). This way, we will have created an edge between the vertex and itself (Vertex1 Vertex2). Since we are creating an additional path, this will allow the algorithm to avoid going through the

same vertices. This way, all the in-edges will be pointing to Vertex1 and all the out-edges will be pointing to V2.
Example of Yes to **A** and No to **B** is shown by figure 1 below. Since a distinct path exists for each vertex X to each vertex in S, this won't hold true for B since each vertex in X must pass through a vertex v1, or v2 in order to reach the vertex S. Due to the pigeonhole principle, since there are only two V vertices, B will not hold true.