16. a) Subtree (x) is defined as the set of vertices that are visited if we call DFS(x) on G before calling DFS() on any other vertex. We will do a proff by contradiction and assume that there exists no such x, such that x has a path to every other vertex.

Now we will choose a vertex w such that subtree (w) has maximum number of vertices, if there are multiple such vertices, we can choose any one arbitarily.

According to our assumption (that there exists no such x) there should be present some vertex v which is not reachable from w. This implies, v is not in subtree (w). Since v is not reachable from w, there must be $v \rightsquigarrow w$, according to the statement of the claim to be proven. Therefore w occurs in subtree (v), which again indicates all vertices present in subtree (w) will also be present in subtree (v). So, subtree (v) is going to have at least one more node than subtree (w). This contradicts our initial assumption that w is such a vertex such that subtree (w) has maximum number of vertices.

Hence proved, There must be some vertex x such that x has a path to every other vertex.

b) The above claim is true even if G is an undirected graph. Because according to the statement given in the claim, that it holds that either $u \dashrightarrow v$ or $v \dashrightarrow u$ or both for every pair (u, v) of vertices, in case of undirected graph it simply becomes the both way path exists for every pair (u, v). Which clearly implies that G is a connected graph. Now, according to the definition of connected graph, there should exist path from every vertex to every other vertex in G, which means here in case of undirected graph G every vertex will be x, such that x has a path to every other vertex.

16. c) Subtree (x) is defined as the set of vertices that are visited if we call DFS(x) on G before calling DFS() on any other vertex. We will do a proof by contradiction and our initial assumption is that for every pair of vertices x and y, either $x \dashrightarrow y$ or $y \dashrightarrow x$ or both.

Now let us choose a vertex w such that subtree (w) has the maximum number of vertices, and if there are multiple such vertices, we can choose any one arbitarily.

According to the statement of the claim, there exists some vertex u that is not reachable from w, which means u is not present in subtree (w). Therefore there must exist a path from u to w, according to our initial assumption. This implies w is present in subtree (u), so every node present in subtree(w) will also be present in subtree (u) plus at least one more node than subtree(w). Therefore subtree (w) remains no more as the subtree with maximum number of vertices, which contradicts our choice of w as the node with largest subtree.

Hence proved, there must be a pair of vertices (x, y) such that $x -!\rightarrow y$ and $y -!\rightarrow x$.

16. d) A graph G is nice if and only if G contains at least one Supersource and G^R i.e., Reverse (G) also contains at least one Supersource.

16. e) ALGORITHM :

```
// returns TRUE if graph G is nice, otherwise returns FALSE.

def isNiceGraph (G) :
    GR = Reverse (G)
    if findSuperSource (G) AND findSuperSource (GR) :
        return TRUE
    else :
        return FALSE

def Reverse (G):          // returns a reversed graph GR
    VR = V                // set of vertices of GR is VR
    ER = ϕ                // set of edges of GR is ER
    for each edge (u,v) in E :
        ER = ER U (v,u)
    return GR (VR, ER)
```

Note: Reverse of a directed graph G(V,E) is another directed graph G_R which contains the same set of vertices V as in G, and same number of edges as in E, but the edges in G_R are in reverse direction, i.e., for every edge (u,v) in G, there exist (v,u) in G_R.

// returns the SuperSource of graph G. If there are more than one supersource present it will simply return the one found first.

```
def findSuperSource(G):
    for each vertex x ∈ V
        V' = ∅
        perform DFS(x) and build V'
        if V' = V:          // check if two vertex set are equal
            return TRUE
    return FALSE
```

Note: here V' is a vertex set which contains all the vertices visited while performing DFS(x) where $x \in V$.

Time Complexity:

let T_1 is the time complexity of Reverse(G) and

T_2 is the time complexity of findSuperSource(G) and

T is the time complexity of isNiceGraph(G)

$$T = T_1 + T_2$$

$T_1 = O(V+E)$ [because we are simply exploring all edges of G and adding its reversed edge to G_R.]

$T_2 = O(V) * O(V+E)$ [because we are performing DFS for every vertex of G]

$T_2 = O(V^2 + VE)$

Therefore $T = O(V+E) + O(V*V + V*E) = O(V^2 + VE)$

Space Complexity: $O(V)$
(because of storing G_R and V' set)