Problem type 1:

Please provide a algorithm for the problem below. Any algorithms given should be computationally efficient (please no brute-forcing). If needed, you may refer to the *Explore* algorithm (below) from lecture as a black box:

```
EXPLORE(G,u):
    Visited[1..n] ← false
    Add u to ToExplore and to S
    Visited[u] ← true
    While (ToExplore is non-empty)
        Remove node x from ToExplore
        for each edge xy in Adj(x)
            if (Visited[y] = false)
                Visited[y] ← true
                Add y to ToExplore
                Add y to S
    return S
```

(See variants below)

a. **BYG**

G is a directed graph and I want to know if node u can reach node v.

b. **BYD**

G is a directed graph and I want to find all nodes that u can reach.

c. **BYA**

G is a directed graph and I want to find all nodes that can reach u .

d. **BYH**

G is a directed graph and I want to find all nodes in u's strong connected component.

Problem type 2:

Answer the following problem:

(See variants below)

a. **BYC**

What type of *directed*-graph has only one strongly connected component?

b. **BYF**

Assuming a directed graph with n nodes, how many edges do you need to make the graph have a single strongly connected component.

c. **BYE**

Assuming a *undirected* graph with n nodes, how many edges do you need to make the graph connected.

d. **BYB**

Assuming a *directed* graph with n nodes, what type of graph would have the most number of cycles of size n. How many cycles would this graph have.