# CS180 Discussion

Week 3

## Lecture Recap

- Topological sort
- Eulerian cycles

## Build Order

Build Order: You are given a list of projects and a list of dependencies (which is a list of pairs of projects, where the second project is dependent on the first project). All of a project's dependencies must be built before the project is. Find a build order that will allow the projects to be built. If there is no valid build order, return an error.

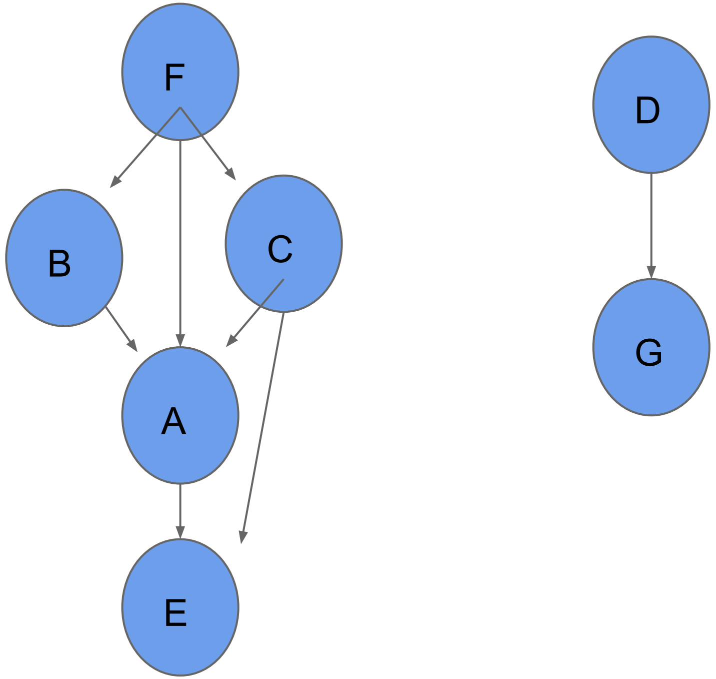
EXAMPLE

Input:

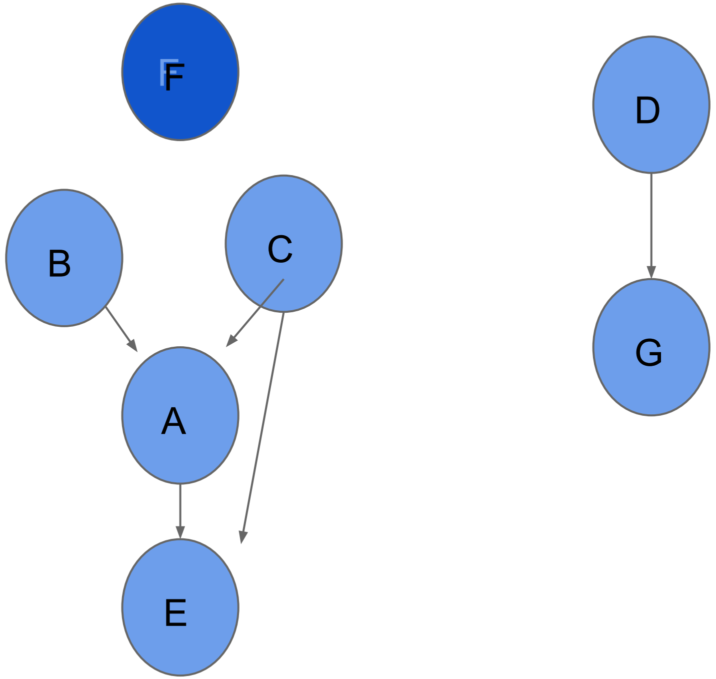
projects: a, b, c, d, e, f

dependencies: (a, e), (f, b), (b, a), (f, c), (d, g) (c, a) (f,a) Output: f, d, c, b, a, g, e

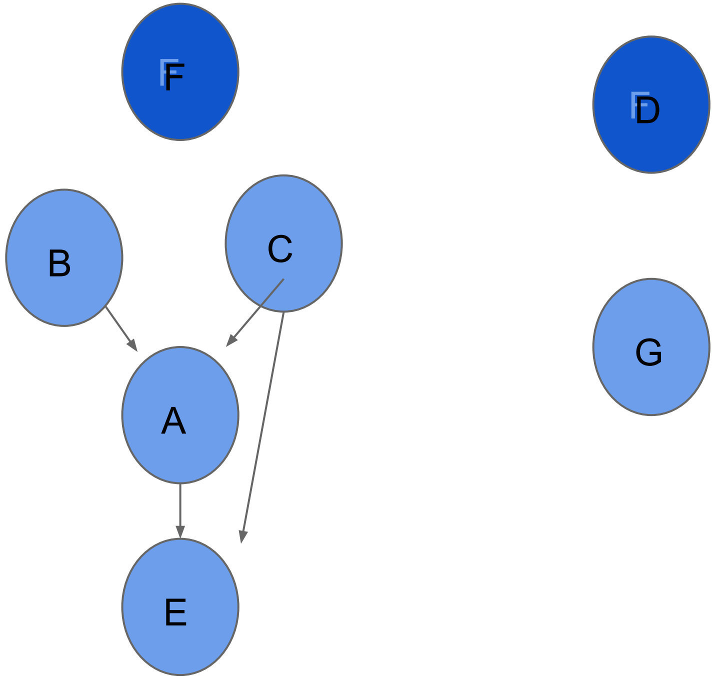
## Build dependencies



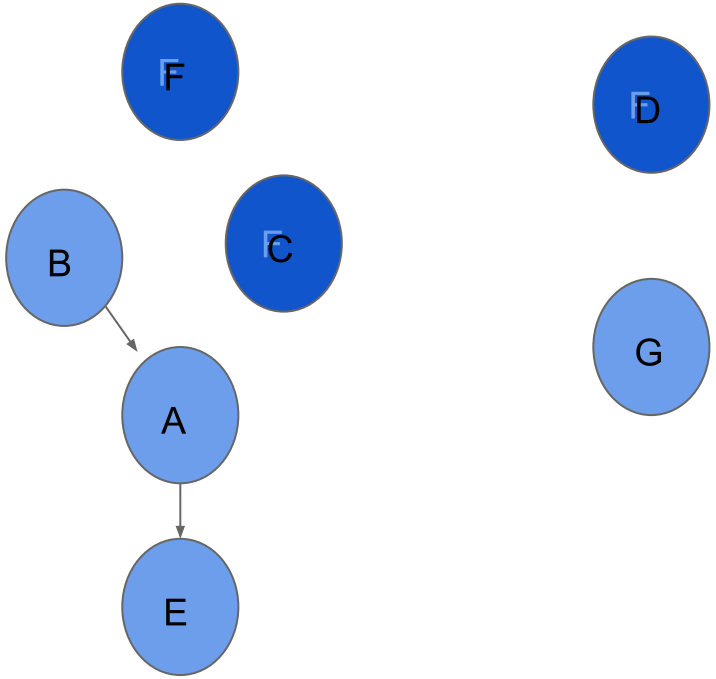
## Build dependencies



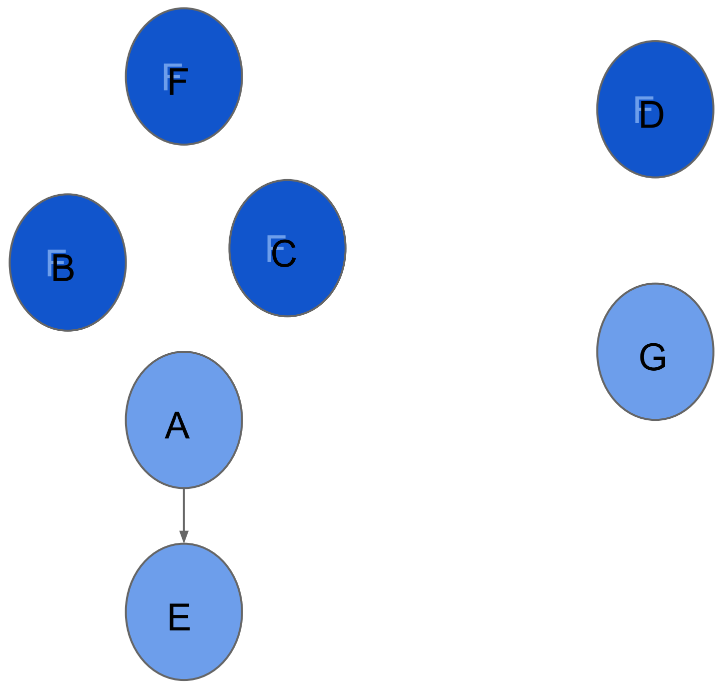
## Build dependencies



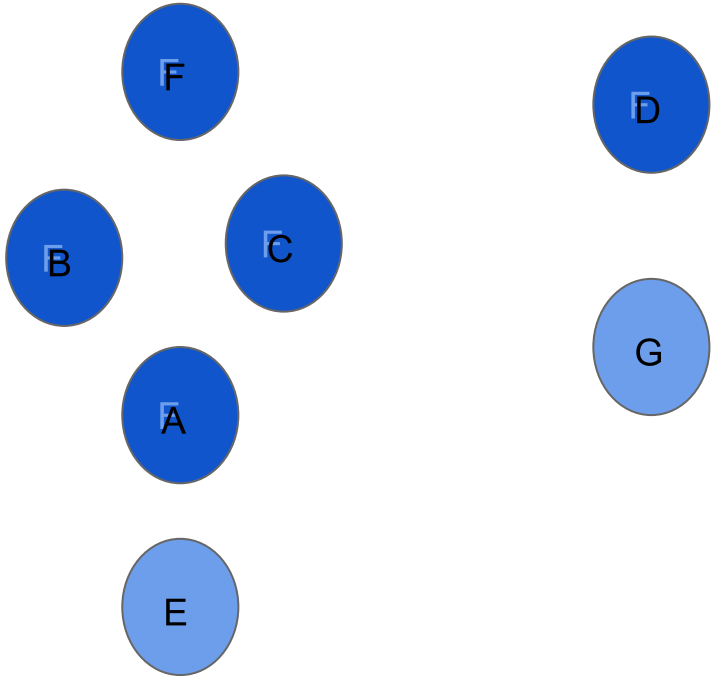
## Build dependencies



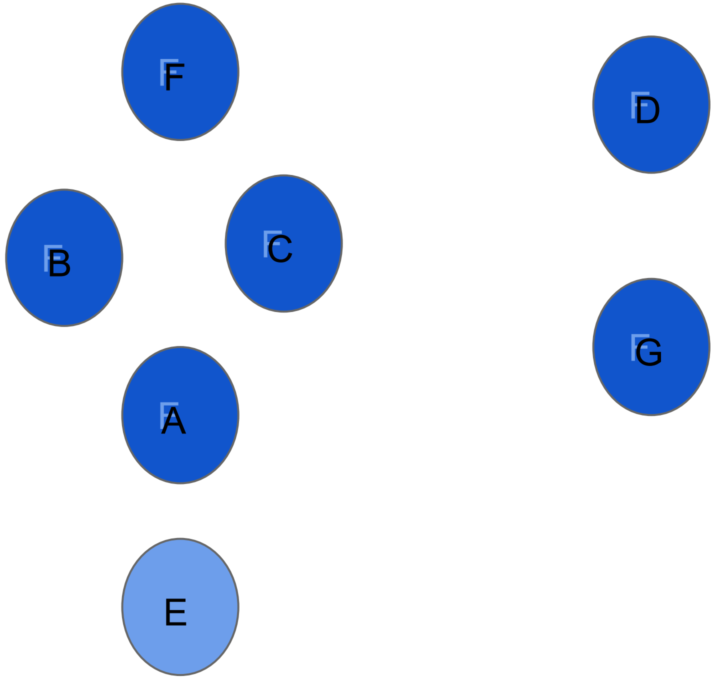
## Build dependencies



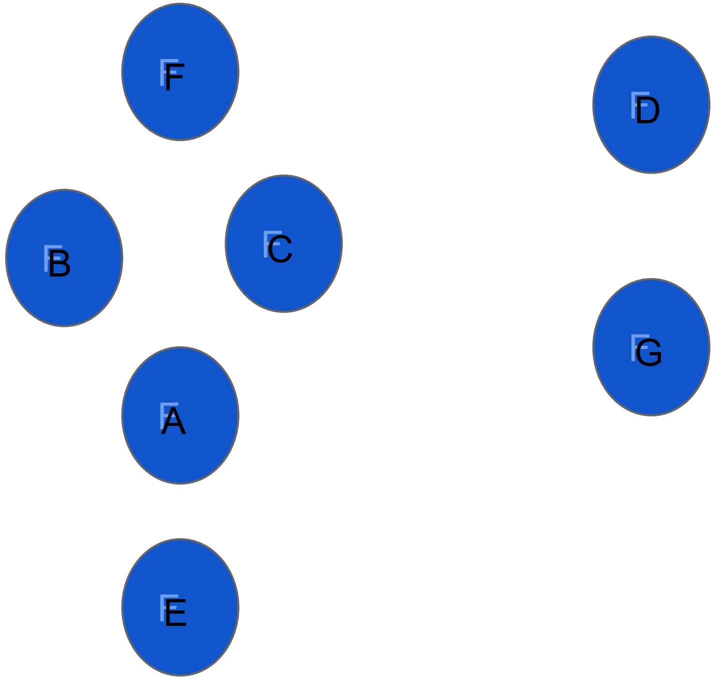
# Build dependencies



## Build dependencies

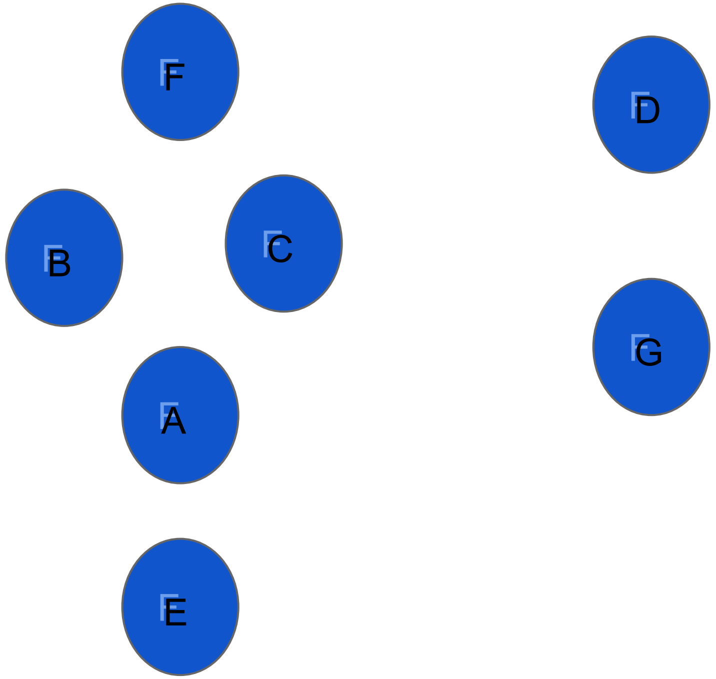


# Build dependencies

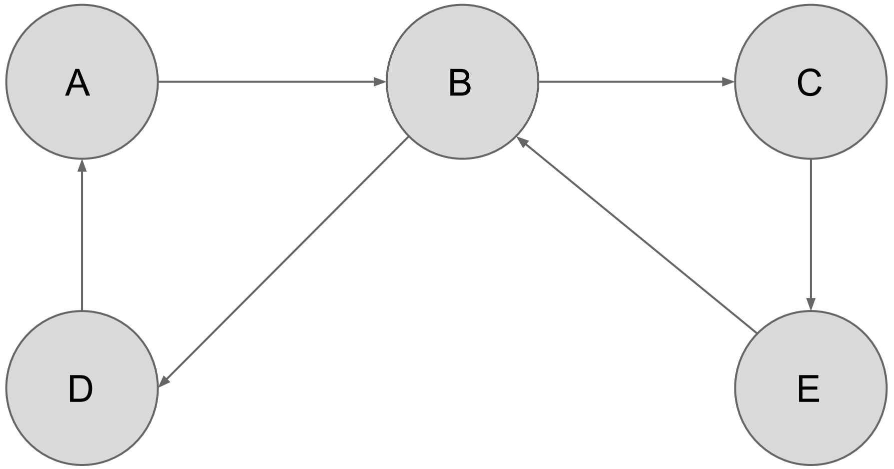


## Build dependencies

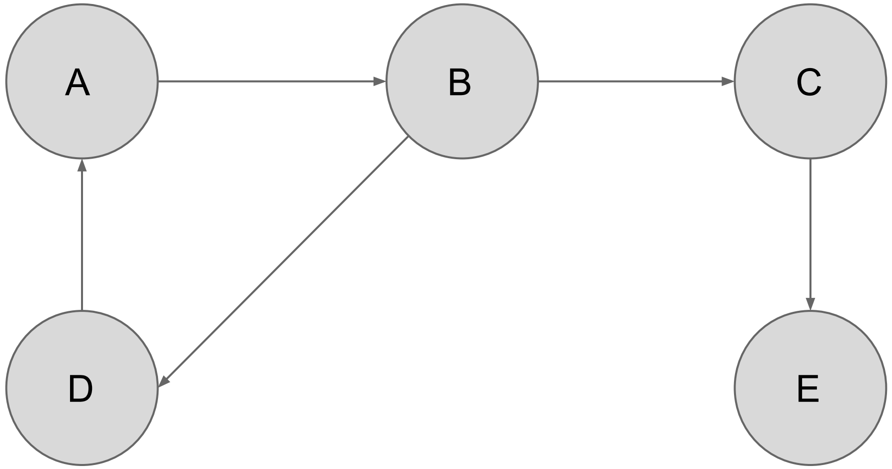
$O(P + D)$



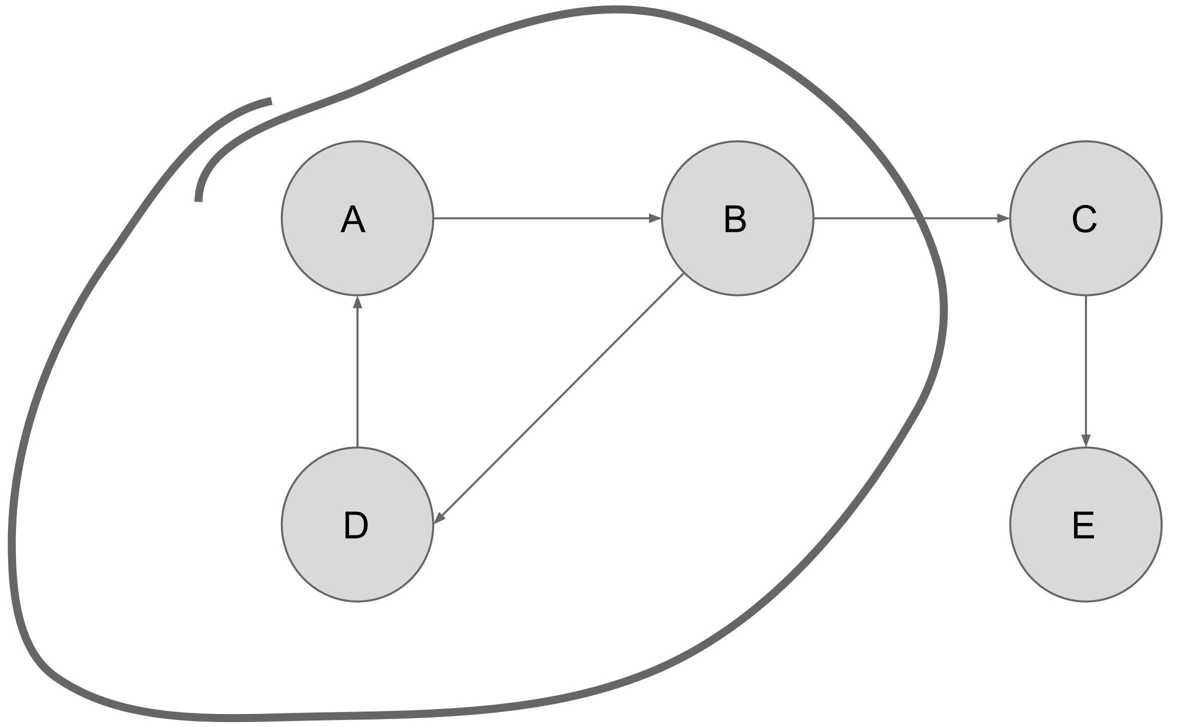
## Strongly connected (?)



# Strongly connected (?)



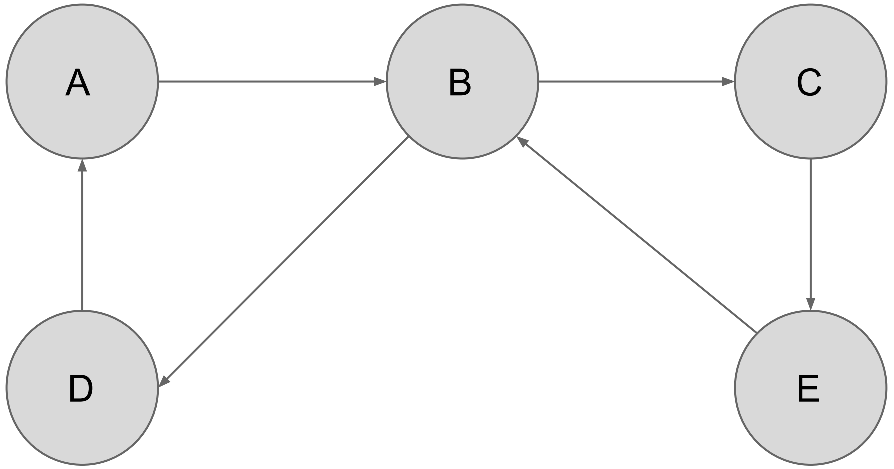
## Strongly connected



# Kosaraju's algorithm

```
L = empty list
For each vertex u of the graph do Visit(u), where Visit(u) is the recursive subroutine:
    If u is unvisited then:
        Mark u as visited.
        For each out-neighbour v of u, do Visit(v).
        Prepend u to L.
    Otherwise do nothing.
For each element u of L in order, do Assign(u,u) where Assign(u,root) is the recursive
subroutine:
    If u has not been assigned to a component then:
        Assign u as belonging to the component whose root is root.
        For each in-neighbour v of u, do Assign(v,root).
    Otherwise do nothing.
```

## Eulerian path



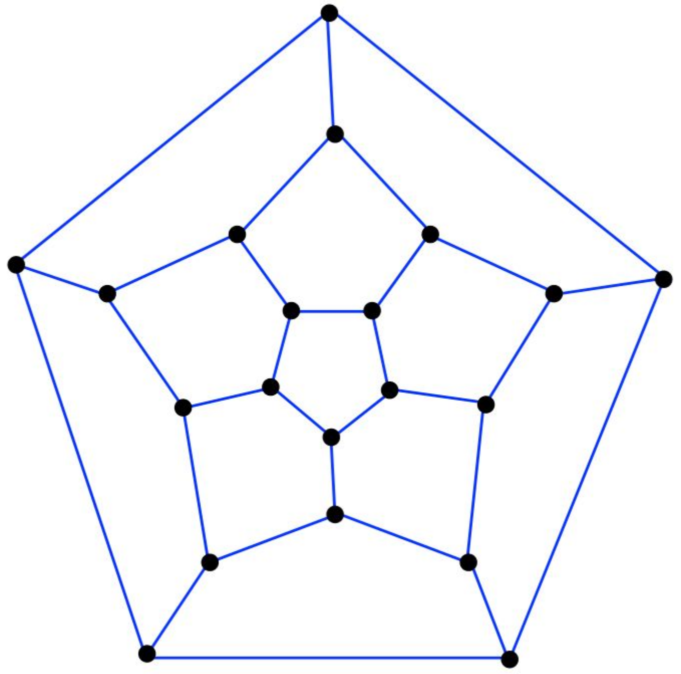
## Eulerian cycle

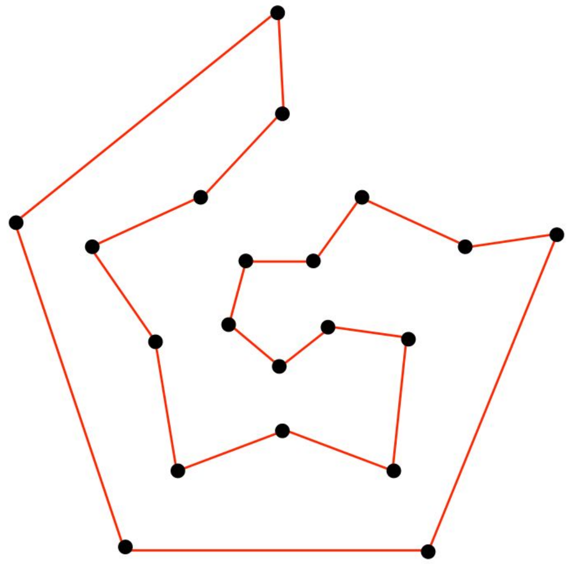
A directed graph has an eulerian cycle if following conditions are true

1) All vertices with nonzero degree belong to a single strongly connected component.

2) In degree and out degree of every vertex is same.

# Hamiltonian path





## Hamiltonian path

A graph has a hamiltonian path if following conditions are true

1) You can find a path that visits each node exactly once.

## Q2.

A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

## Q2. solution

- A tree with a single node has 0 children.
- The number of nodes with two children is 0.

a. Adding a node to an existing node that has no children, does not change the number of nodes with two children, nor the number of leaves.
b. Adding a node to an existing node that has one child, increases the number of nodes with two children by 1, and also increases the number of leaves by 1.

Since the number of nodes with two children starts as exactly one less than the number of leaves, and adding a node to the tree either changes neither number, or increases both by exactly one, then the difference between them will always be exactly one.

terview Questions!!!

Interview Questions!!!

Interview Questions!!!

Interview Questions!!!

## Detecting Loops

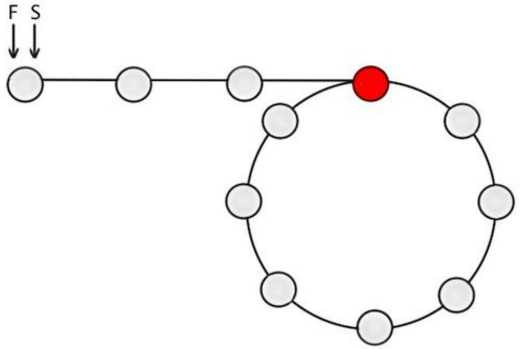
Loop Detection: Given a circular linked list, implement an algorithm that returns the node at the beginning of the loop.

DEFINITION:

Circular linked list: A (corrupt) linked list in which a node's next pointer points to an earlier node, so as to make a loop in the linked list.

EXAMPLE

Input: A -> B -> C -> D -> E -> C    [the same C as earlier]    Output: C



we move FastPointer twice as fast as SlowPointer. When SlowPointer enters the loop, after k nodes, FastPointer is k nodes into the loop. This means that FastPointer and SlowPointer are LOOP_SIZE - k nodes away from each other.
Next, if FastPointer moves two nodes for each node that SlowPointer moves, they move one node closer to each other on each turn. Therefore, they will meet after LOOP_SIZE - k turns. Both will be k nodes from the front of the loop.
The head of the linked list is also k nodes from the front of the loop. So, if we keep one pointer where it is, and move the other pointer to the head of the linked list, then they will meet at the front of the loop.