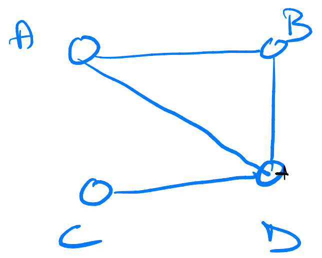
CS II Lecture 4

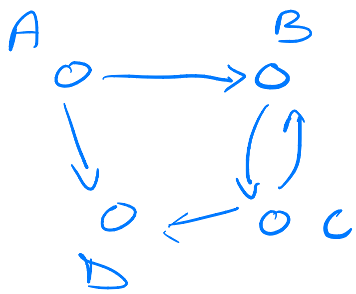
Graphs

- Representations
- Connectivity
- DFS
- Topological Sort

Represent a graph as a matrix



$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \\ B & \\ C & \\ D & \end{array}\quad \begin{array}{c} A \\ B \\ C \\ D \end{array}\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

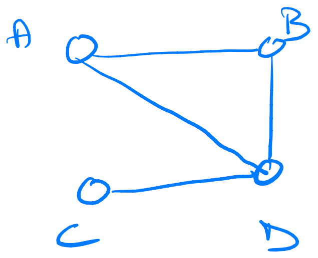


$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$n =$ # vertices

- check if two given vertices are joined by an edge: time $O(1)$
- list neighbors of a given vertex: time $O(n)$
- memory use $O(n^2)$
  n^2 bits

Adjacency List Representation

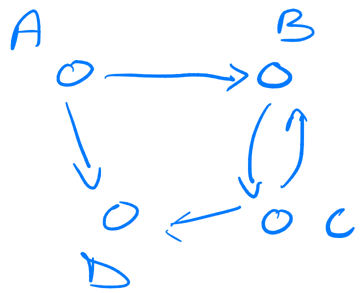


A : B, D

B : A, D

C : D

D : A, B, C



A : B, D

B : C

C : B, D

D : $\emptyset$

n = # vertices m = # edges degree(v) = # neighbors of v

Check if edge (u, v) exists: time $O(\text{degree}(u))$ ✗

List neighbors of v: time $O(\text{degree}(v))$ ✓

Memory used: $\sum_v O(\text{degree}(v)) = O(m)$ ✓

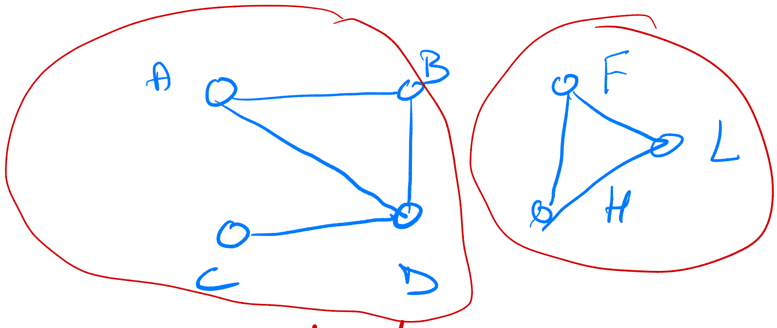
undirected

$$\sum_v \text{degree}(v) = 2m$$

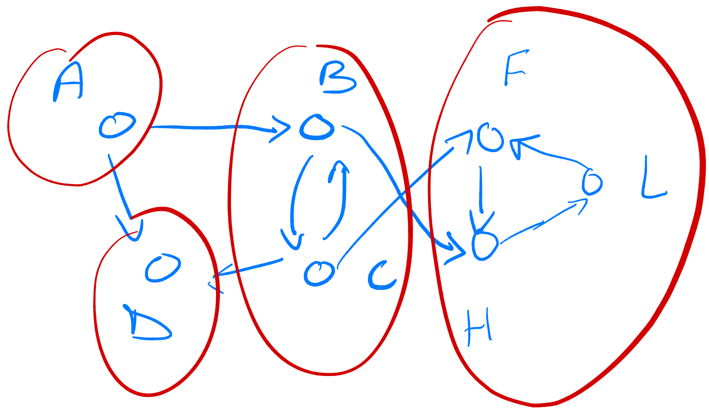
directed

$$\sum_v \text{degree}(v) = m$$

Connected Components



Strongly Connected Components



Depth-First Search

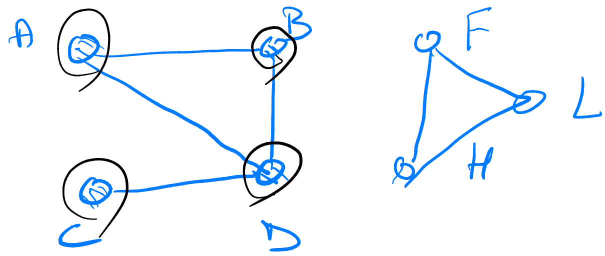
Global variables:

graph $G = (V, E)$

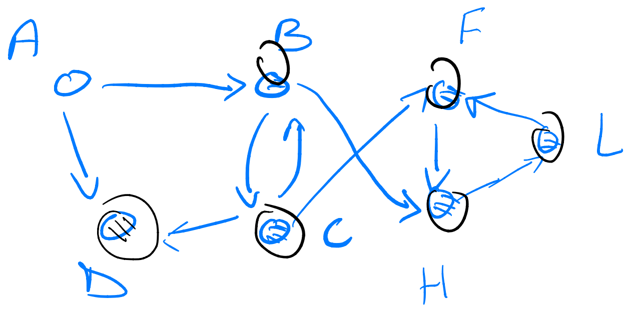
boolean array visited[ ] one entry per vertex initialized to F

```
def explore(v):
    visited[v] = T
    for each w such that (v,w) ∈ E:
        if not visited[w]:
            explore(w)
```

```
def DFS():
    for each v in V:
        if not visited[v]:
            explore(v)
```



explore (C)
explore (D)
explore (A)
explore (B)



explore (C)

Connected Components

Global variables:

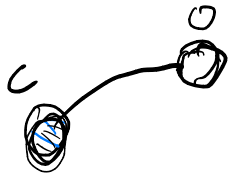
graph $G = (V, E)$

array visited[ ] one entry per vertex initialized to F

array comp[ ] of integers, one entry per vertex

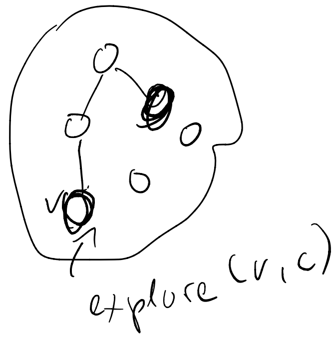
```
def explore(v, c)
    visited[v] = T ; comp[v] = c
    for each w such that (v,w) ∈ E :
        if not visited[w]:
            explore(w, c)
```

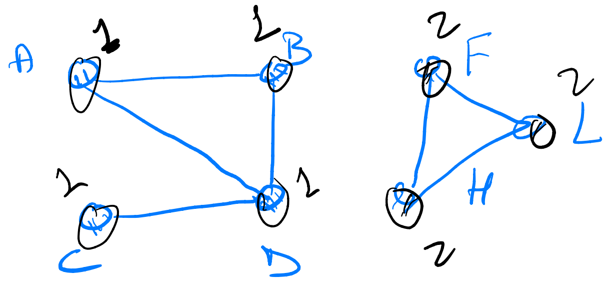
```
def DFS_cc()
    c = 0
    for each v in V:
        if not visited[v]:
            c = c + 1
            explore(v, c)
```



comp 1 1 1 1 2 2 2

A B C D F H L

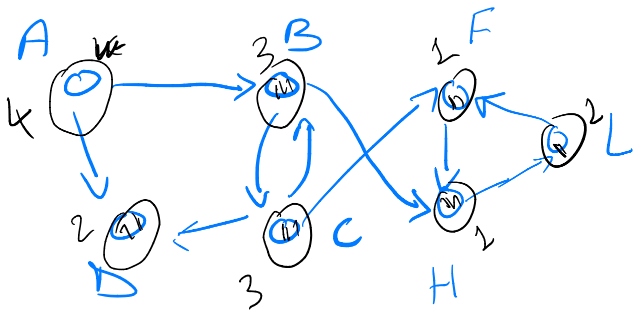




DFS-CC

explore (a, 1)

explore (F, 2)



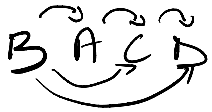
DFS-CC

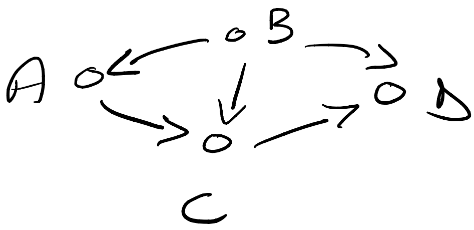
if nodes are considered in order

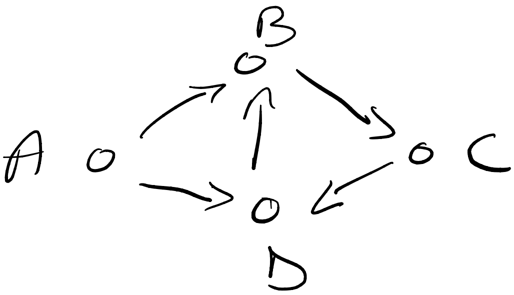
- A, B, C, D, F, H, L

- L, D, B, C, A, F, H

Topological Sort of Directed Graphs







Topological Sort and Directed Cycles

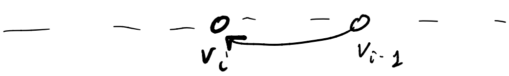
- If a directed graph has a cycle

$$v_1 \to v_2 \to v_3 \to \ldots \to v_k \to v_1$$

$(v_1, v_2) \in E$
$(v_2, v_3) \in E$
$\vdots$
$(v_{k-1}, v_k) \in E$
$(v_k, v_1) \in E$

Then graph has no topological sort

- suppose there was one
  consider first vertex of
  cycle in the order of
  the top. sort

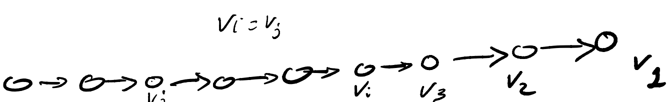


- If a directed graph has no cycle then it has a t.s.

Consider algorithm:

takes a vertex with no incoming edge
puts it first in ordering
removes from graphs
continue recursively

$\boxed{\text{Claim: if a graph is such that every vertex has} \geq 1 \text{ incoming edge, then the graph has a cycle}}$



Topological Sort and DFS

Global variables:

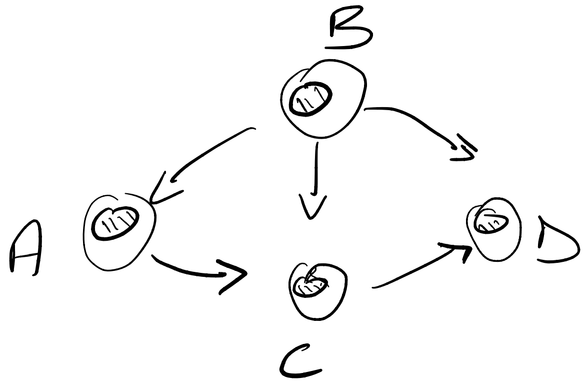
graph $G = (V, E)$

array visited[ ] one entry per vertex initialized to F

list L initialized to $\emptyset$

```
def explore(v):
    visited[v] = T
    for each w such that (v,w) ∈ E:
        if not visited[w]:
            explore(v)
    add v at head of L
```

```
def DFS():
    for each v in V:
        if not visited[v]:
            explore(v)
```



explore(A)
explore(C)
explore(D)
explore(B)

L = BACD

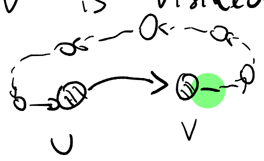
Lemma

Suppose G is a Directed Acyclic Graph and (u,v) is an edge.
Then explore(v) terminates before explore(u) terminates
And so u will be placed before v in L

Proof

when run explore(u) and look at (u,v)

v is visited

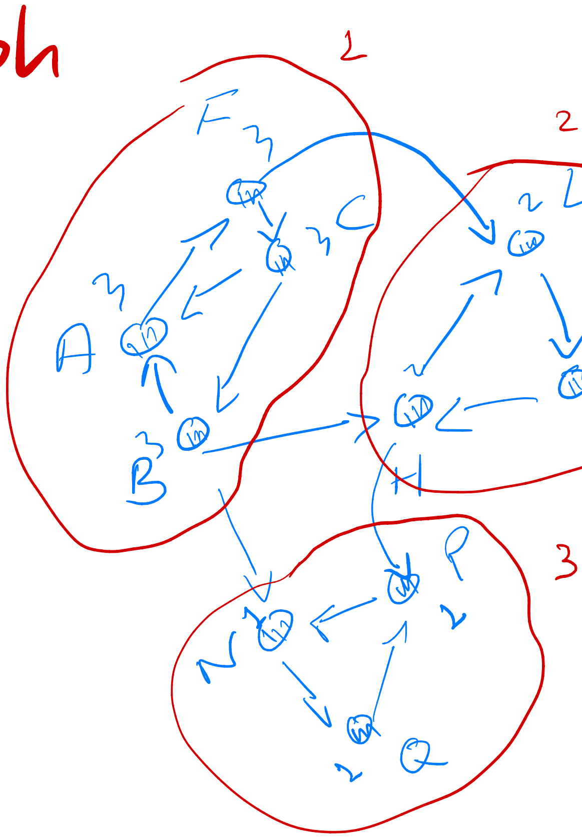


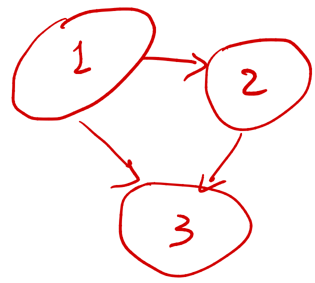
explore(v) has terminated
explore(u) still running

v is not visited

explore(u)
explore(v)
⋮

Strongly Connected Components of a Directed Graph

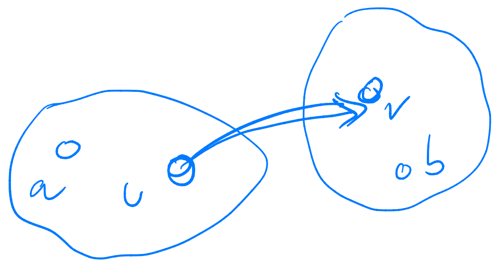




1, 2, 3

N R Q L H M A B F C

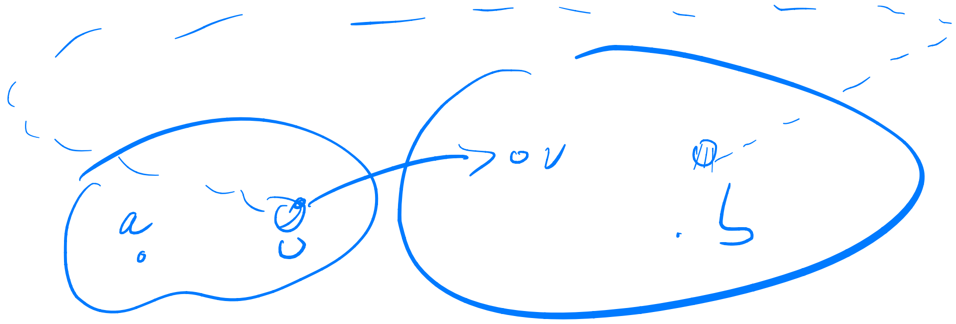
Lemma



explore(u)

v visited | v not visited

If (u,v) is an edge, a is last node for which explore() terminates in component of u, b is last node for which explore() terminates in component of v, explore(b) terminates before explore(a) terminates



explore (u)

considers (u, v)

b not visited

explore

b visited

Algorithm for strongly connected components

Given directed graph G

- construct graph G^R obtained by reversing each edge of G
- run topological sort algorithm on G^R, obtain list L
- run undirected connected component algorithm on G, using L as ordering in "for each v" in procedure "DFS"