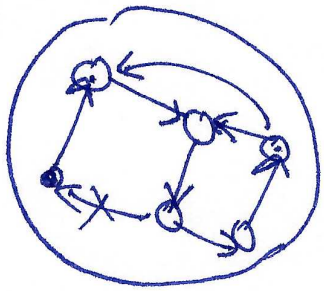
DFS (Depth-first search)



```
DFS(s)
{
   for every u ∈ N(s) that hasn't been visited
   {
      visit u
      DFS(u)
   }
}
```

```
BFS(s)
{
   Start with empty queue Q
   enqueue(s)
   while (Q is non-empty)
   {
      u = dequeue(Q)
      for every v ∈ N(u) --
      {
         enqueue
      }
   }
```

DFS(s)

Replace the queue with a stack

---

pred
visited
distance
} Array

Init.

pred[v] is NULL for all v ∈ V

visited[v] = false

Arrays { pred, visited, disc, finish

time = 0

Tree

Boolean finds connected component

DFS(s)

(0) visited[s] = true

(1) disc[s] = time

(2) time++

(3) for all $u \in N(s)$: ← out-neighborhood

  (a) If visited[u] is false:

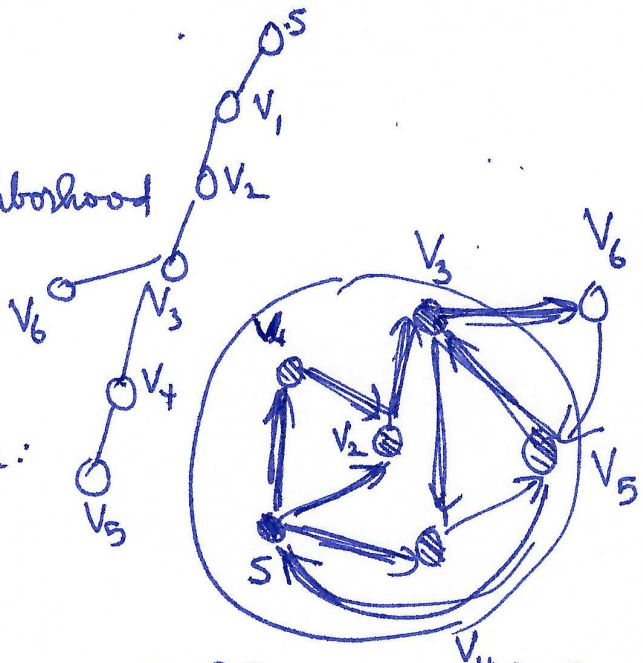
    (i) visited[u] = true

    (ii) pred[u] = s

    (iii) DFS(u)

(4) finish[s] = time

(5) time++



disc[s] = 0	fin[v_0] = 11
disc[v_1] = 1	fin[v_1] = 10
disc[v_2] = 2	fin[v_2] = 9
disc[v_3] = 3	fin[v_3] = 8
disc[v_4] = 4	fin[v_4] = 7
disc[v_5] = 5	fin[v_5] = 6

DFS(G)

(1) Initialize all arrays (pred, visited, disc, finish)

(2) time = 0

(3) For all $s \in V$: ← all vertices

  (a) If visited[s] is false

    (i) DFS(s)

If G is undirected, connected components

Output of DFS(G)

(i) pred (DFS forest)

(ii) disc, finish times

Running time = $\Theta(m+n)$

Suppose DFS(u) is a recursive call made by DFS(s).

Q. IS $disc[s] < disc[u]$?

(A) Y (B) N

Q. Is $finish[s] < finish[u]$?

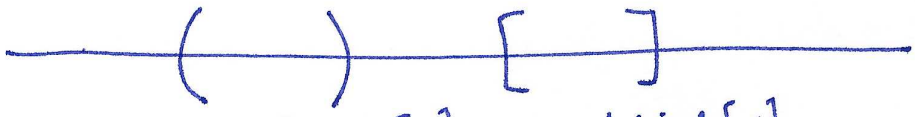
(A) Y (B) N (C) Not sure

Thm [Parentheses Thm]:

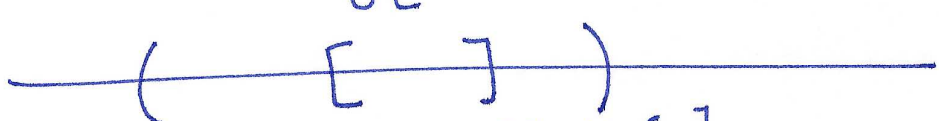
u v $disc[u] < disc[v]$

Suppose

Then



or



NOT

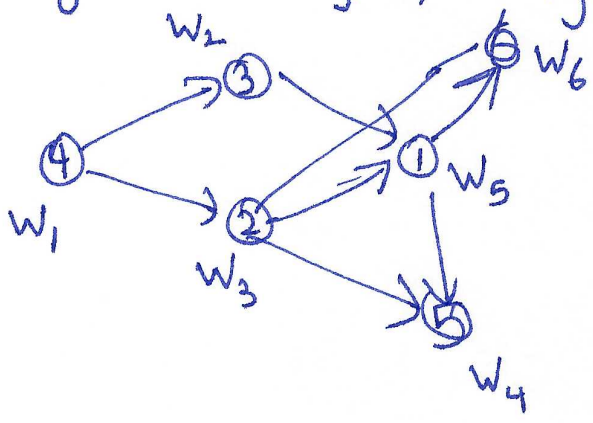
DAG (Directed Acyclic Graph)

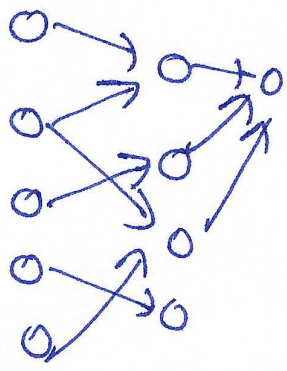
(1) Given a directed graph G, determine if G is a DAG.

(2) Given DAG G, find a <u>topological ordering</u> of vertices.

Order vertices as $w_1, w_2, \ldots, w_n$. st

for all edges (w_i, w_j), $i < j$

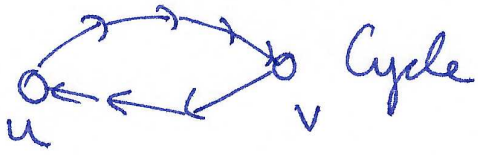




$u \rightsquigarrow v$ as a shorthand for "there is a directed path from u to v"

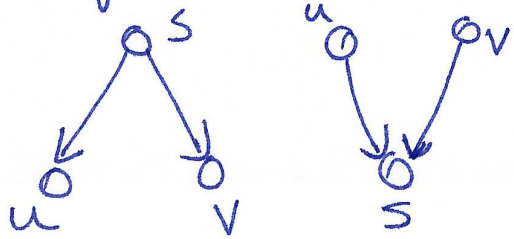
Q. If G is a DAG, can both $u \rightsquigarrow v$ and $v \rightsquigarrow u$ be true?

(A) Y ✓ (B) N

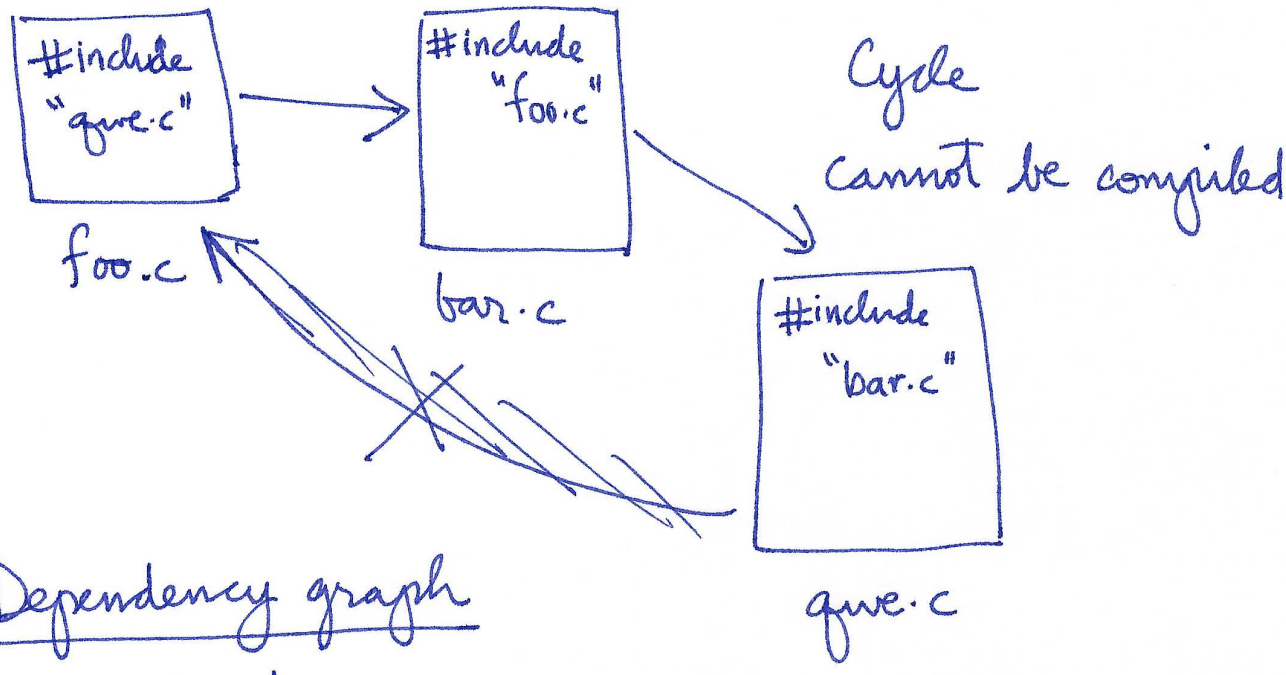


Q. If G is a DAG, does one of $u \rightsquigarrow v$ and $v \rightsquigarrow u$ have to be true?

(A) Y ✓ (B) N



Topological Sort

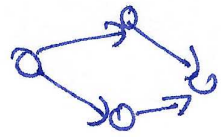


Dependency graph

$G = (V, E)$

V = collection of files

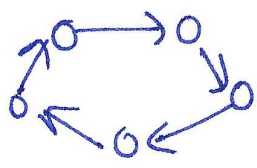
(u, v) is an edge if v "#includes" u



A (directed) cycle is a sequence of vertices

$v_1, v_2 - - -, v_k$ s.t. $\forall\ 1 \le i \le k-1$, (v_i, v_{i+1}) is an edge

AND (v_k, v_1) is an edge.



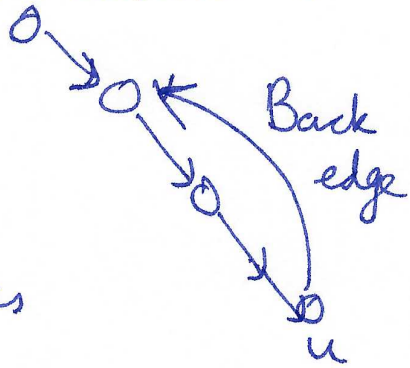
Code cannot be compiled

A "proper" dependency graph should have NO directed cycles.

Run DFS

Def: Back edge of DFS forest

(u,v) is a back edge if u is a descendant of v in DFS forest



If G is a DAG, then can be no back edges

[Thm, that needs a proof]

If there is no back edge, G is a DAG.

Algorithm for determining if G is a DAG

(1) Run DFS(G) $O(m+n)$

(2) for every edge, check if it is back edge (looking at finish times / disc times)

Topological Sort

G is a DAG

Thm: The reverse ordering of finish times of DFS(G) is a topological ordering.

(In DFS, when a vertex finishes, put it at beginning of a list.) → Become topological ordering