Introduction to Algorithms

CS 430

Lecture 21-22

ILLINOIS INSTITUTE OF TECHNOLOGY

By Lan Yao

Outlines

- Terminology of Graph
- Breadth First Search
- Depth First Search

Graph Terminology

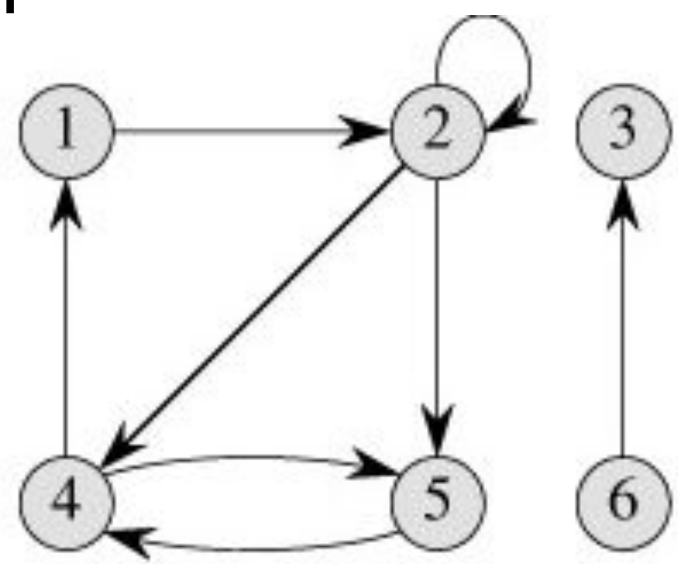
- Graph – set of vertices, set of edges
- Directed Graphs
- Undirected Graphs
- Weighted Graphs
- Adjacent Vertices, Paths

Directed Graphs (Digraphs)

- G is a pair (V, E), where V is a finite set and E is a binary relation on V
- The set V is called the vertex set of G, and its elements are vertices .
- The set E is the edge set of G, and its elements are edges.
- Note that self-loop edges from a vertex to itself are possible.

Directed Graphs (Digraphs)

- A directed graph G = (V, E), where V = {1, 2,3, 4, 5,6} and E = {(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)}. The edge (2, 2) is a self-loop.

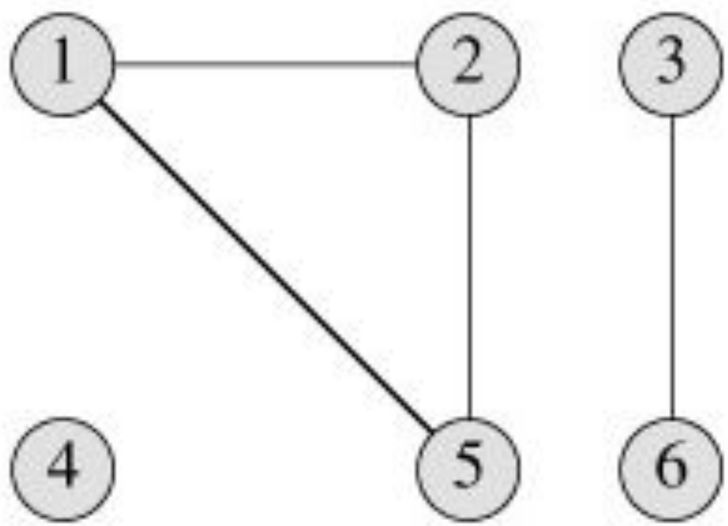


Undirected Graphs

- $G = (V, E)$, the edge set E consists of unordered pairs of vertices, rather than ordered pairs. That is, an edge is a set $\{u, v\}$, where $u \in V$, $v \in V$ and $u \,!= v$.
- By convention, we use the notation (u, v) for an edge, and (u, v) and (v, u) are considered to be the same edge.
- In an undirected graph, self-loops are forbidden, and so every edge consists of exactly two distinct vertices.

Undirected Graphs

- An undirected graph G = (V, E), where V = {1, 2 3, 4, 5, 6} and E = {(1, 2), (1, 5), (2, 5), (3, 6)}. The vertex #4 is isolated.

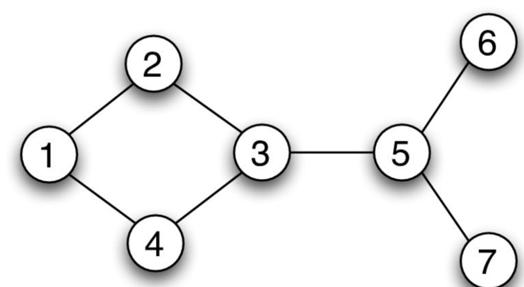


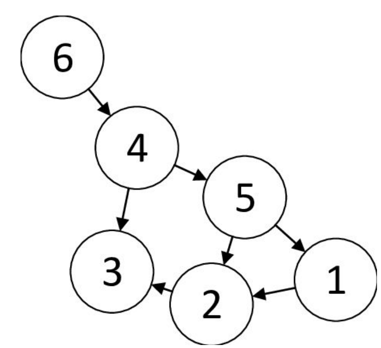
Graph terminology (1)

- If (u, v) is an edge in a directed graph G = (V, E), we say that (u, v) is incident from or leaves vertex u and is incident to or enters vertex v.
- If (u, v) is an edge in an undirected graph G = (V, E), we say that (u, v) is **incident** on vertices u and v.
- If (u, v) is an edge in a graph G = (V, E), we say that vertex v is **adjacent** to vertex u. If the graph is undirected, the adjacency relation is symmetric. If the graph is directed, adjacency relation is not necessarily symmetric. If v is adjacent to u in a directed graph, we sometimes write as **u** → **v**.

Graph terminology (2)

- The **degree** of a vertex in an undirected graph is the number of edges incident on it. A vertex whose degree is 0 is isolated.
- In a directed graph, the **out-degree** of a vertex is the number of edges leaving it, and the **in-degree** of a vertex is the number of edges entering it. The degree of a vertex in a directed graph is its in-degree plus its out-degree.





Graph terminology (3)

- **The length of the path** is the number of edges in the path. If the path contains the vertices v_0, v_1,..., v_k and the edges (v_0, v_1), (v_1, v_2),..., (v_{k-1}, v_k), its length is k.
- There is always a 0-length path from u to u.
- If there is a path p from u to v, we say that **v is reachable from u** via p, which we sometimes

  write as u $\sim>$ v if G is directed.

Graph Algorithms

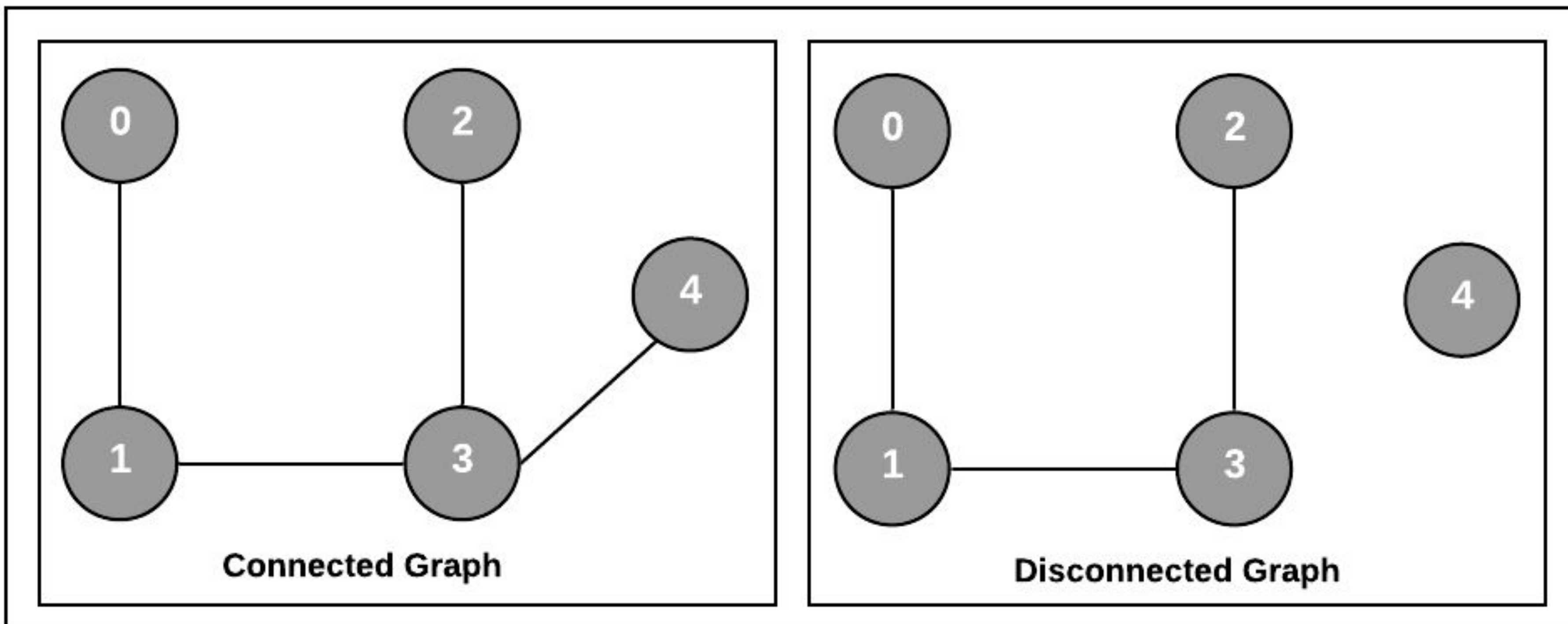
- Finding Cycles
- Connected
- Traversals – Breadth First, Depth First
- Topological Sort
- Strongly Connected Components

Graph terminology (4)

- In a directed graph, a path $<v_0, v_1, \ldots, v_k>$ forms a **cycle** if $v_0 = v_k$ and the path contains at least one edge. A self-loop is a cycle of length 1. A directed graph with no self-loops is **simple**.
- In an undirected graph, a path $<v_0, v_1, \ldots, v_k>$ forms a **cycle** if $k >= 3$, $v_0 = v_k$, and $v_1, v_2, \ldots, v_k$ are distinct.
- A graph with no cycles is **acyclic** (if connected, then a tree).

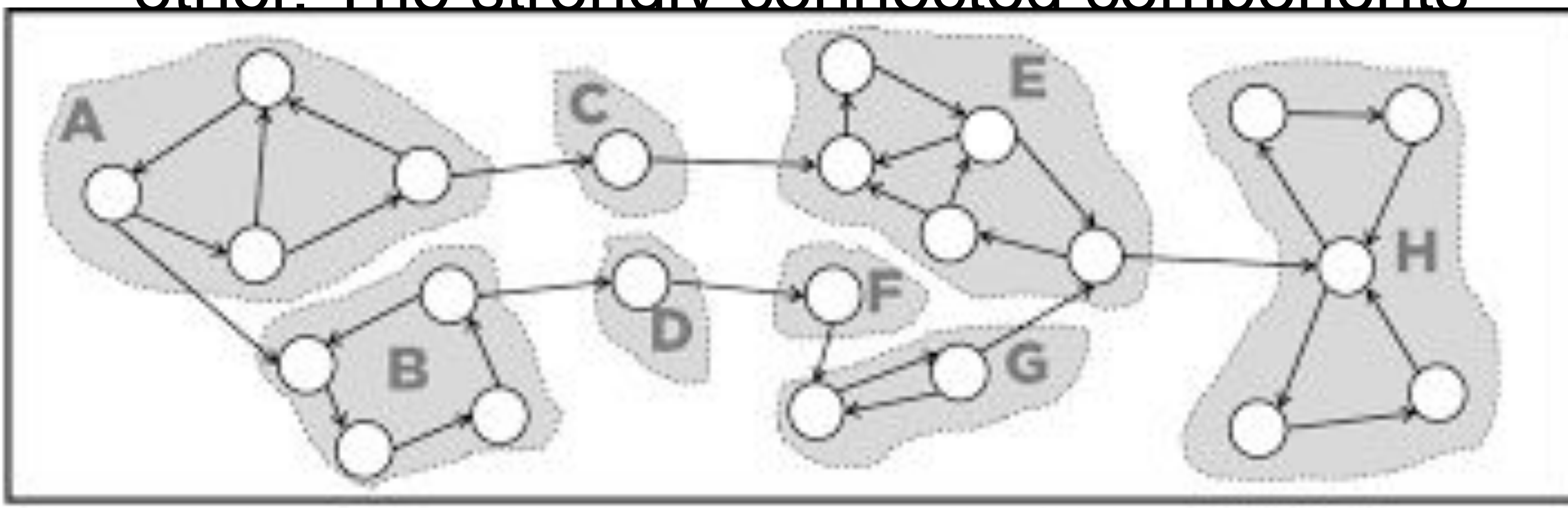
Graph terminology (5)

- An undirected graph is **connected** if every pair of vertices is connected by a path. The connected components of a graph are the



Graph terminology (6)

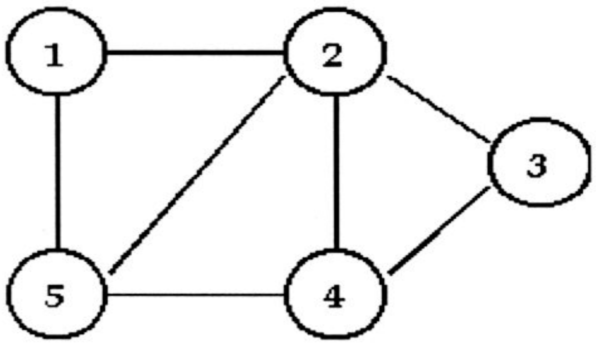
- A directed graph is **strongly connected** if every two vertices are reachable from each other. The strongly connected components

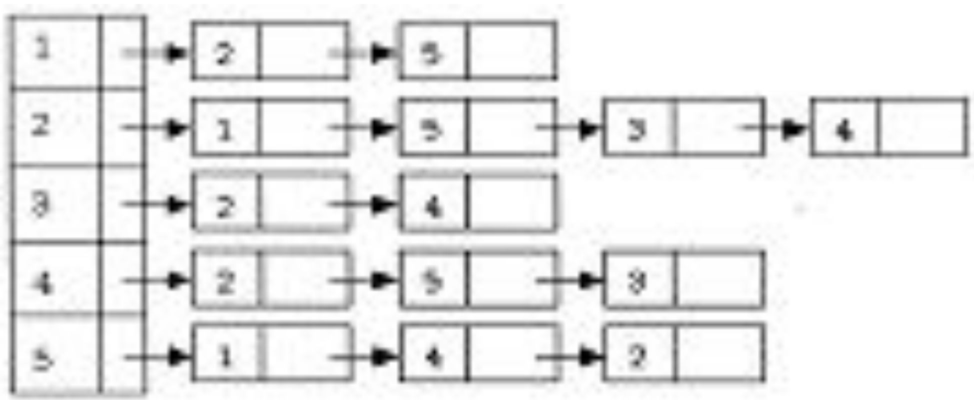


strongly connected components (SCC)

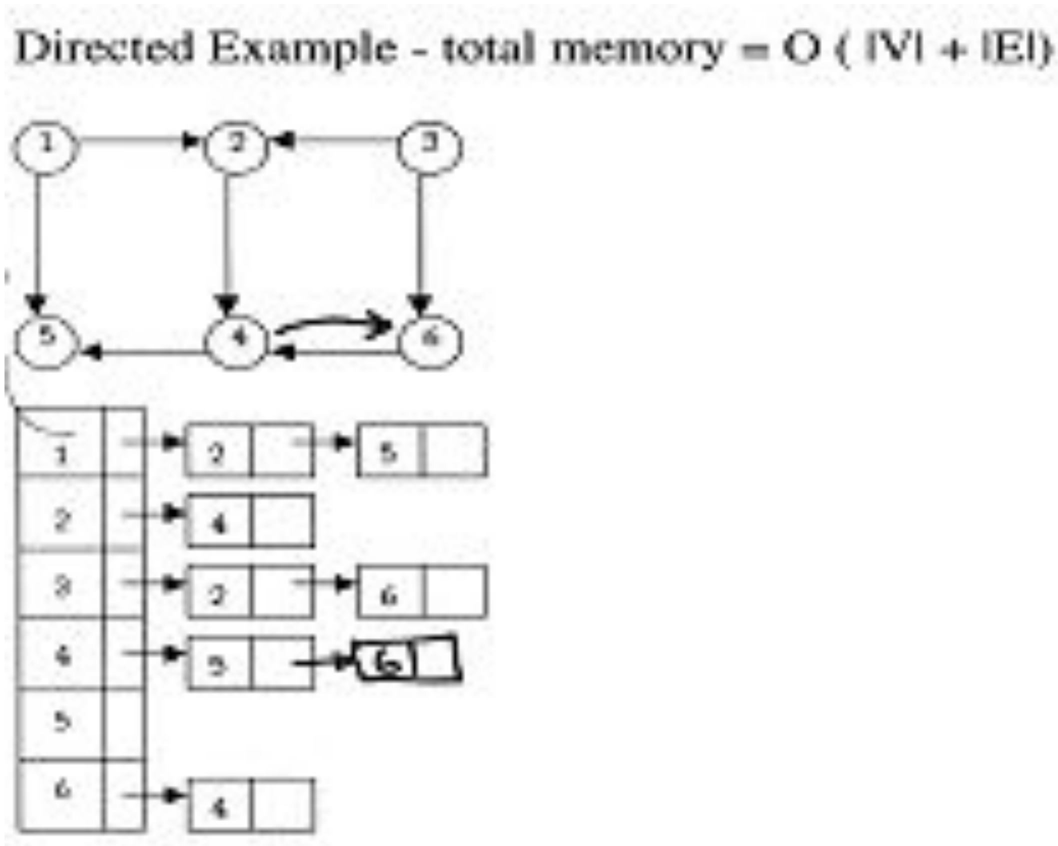
ADJACENCY LIST REPRESENTATION

Undirected Example - total memory = O (|V| + 2*|E|)



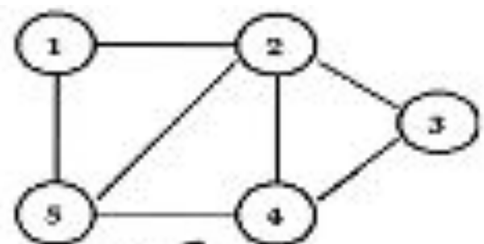


ADJACENCY LIST REPRESENTATION*



ADJACENCY MATRIX REPRESENTATION*

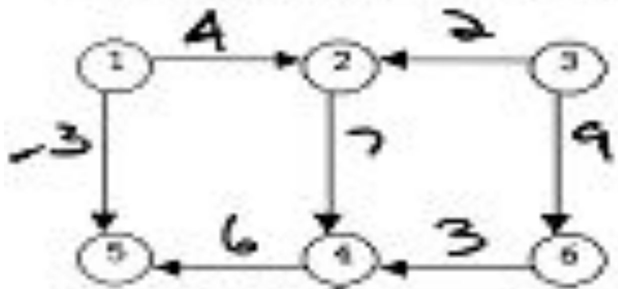
- Better for dense graphs, or for quick knowledge about connections (and paths)
- |V| x |V| matrix
- a(i,j) = 1 if edge exists between vertex i and vertex j, otherwise a(i,j) = 0
- V^2 Memory



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Undirected Example - symmetry required along main diagonal

Directed Example - symmetry not required along main diagonal



	1	2	3	4	5	6
1	0	4	0	0	-3	0
2	0	0	0	1	0	0
3	0	1	0	0	0	1
4	0	0	0	0	1	0
5	0	0	0	0	0	0
6	0	0	0	1	0	0

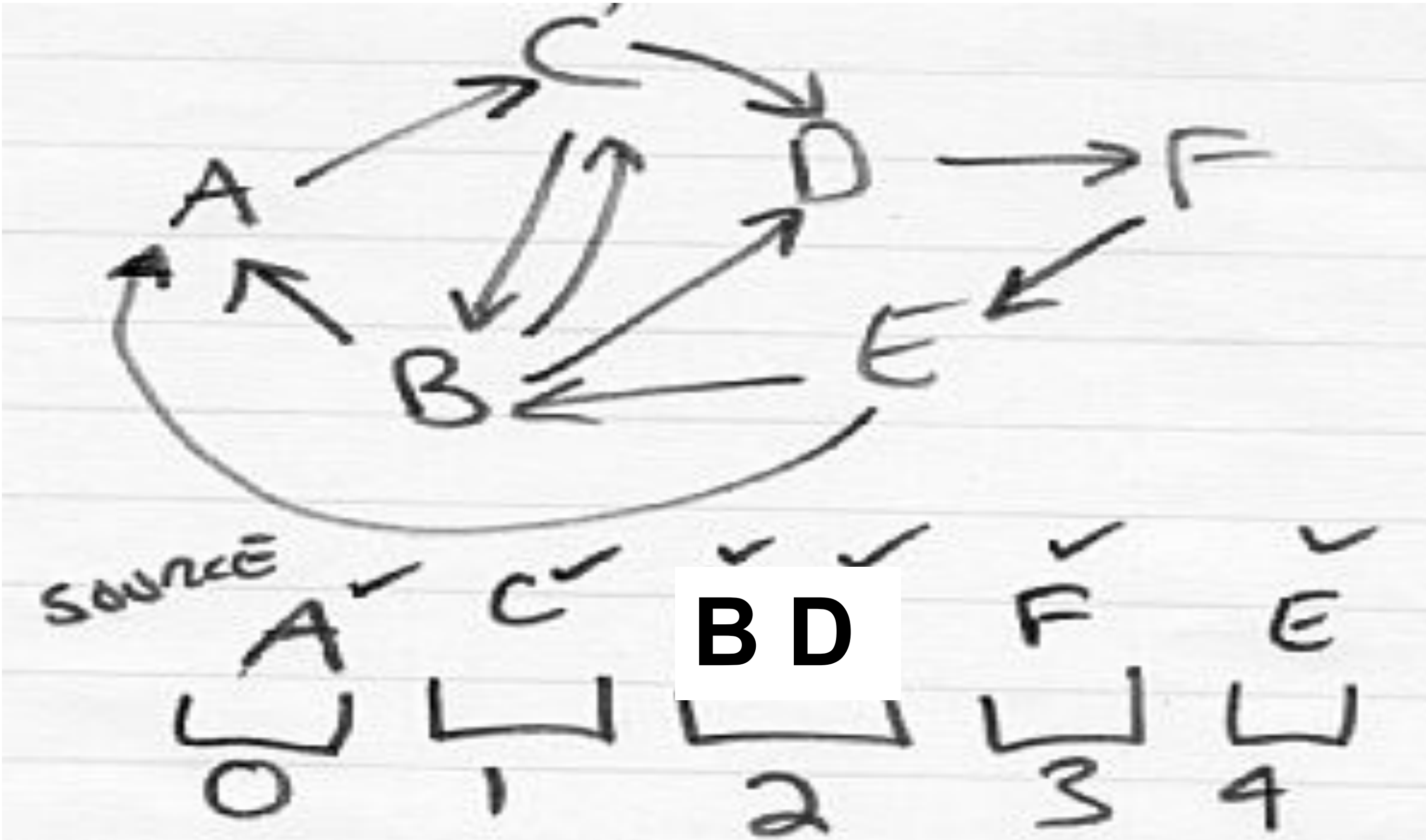
Weighted Graphs

- edges have weights associated with them (length, cost, etc.)
- adjacency lists can store weights of edges in linked nodes

Graph Traversals

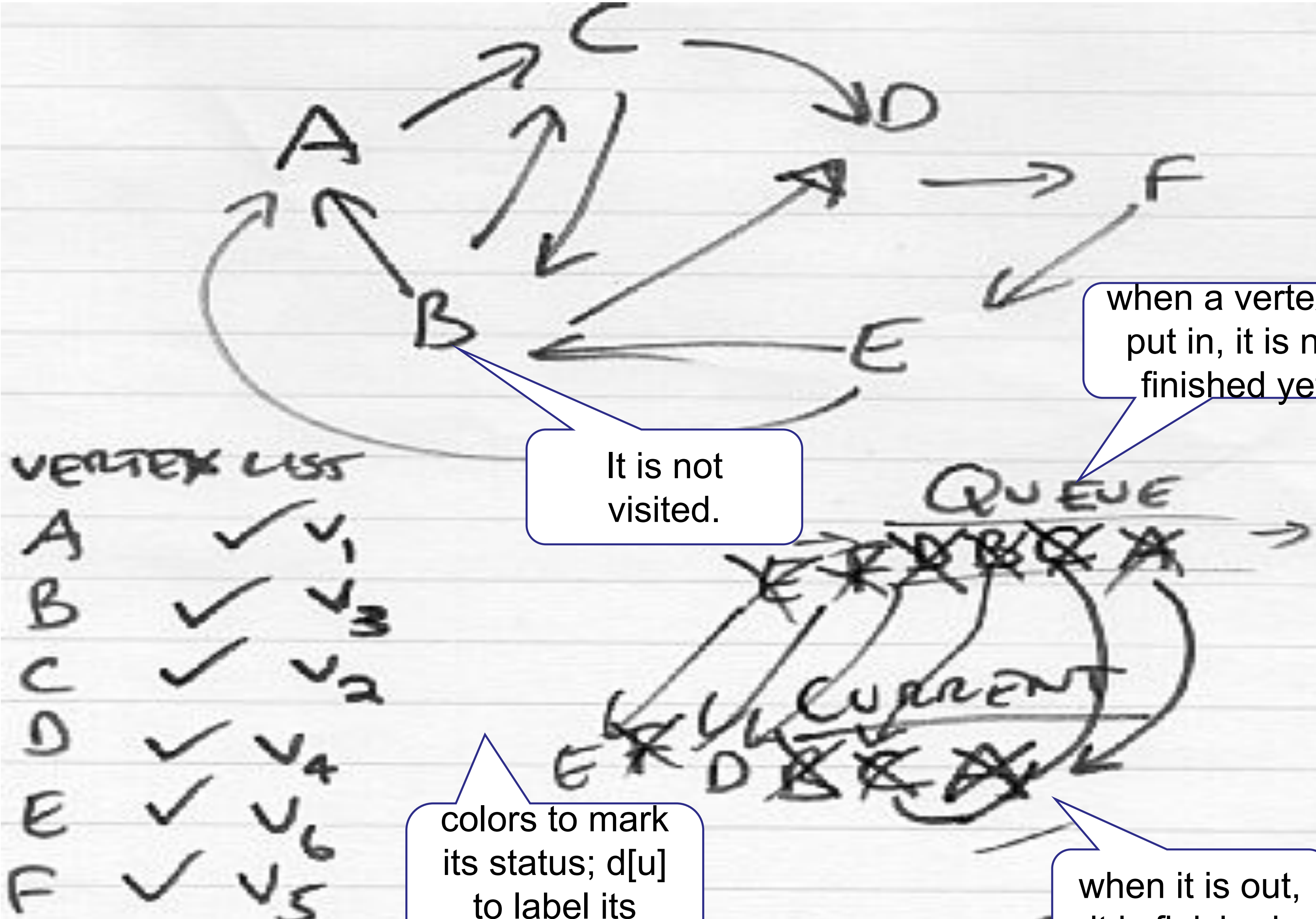
Search / visit vertices in a graph

Breadth first - visit vertices one edge from a given source, two edges from source, etc



Undirected graph - if connected, all vertices will be visited

Directed graph - Must be strongly connected to be able to visit all vertices

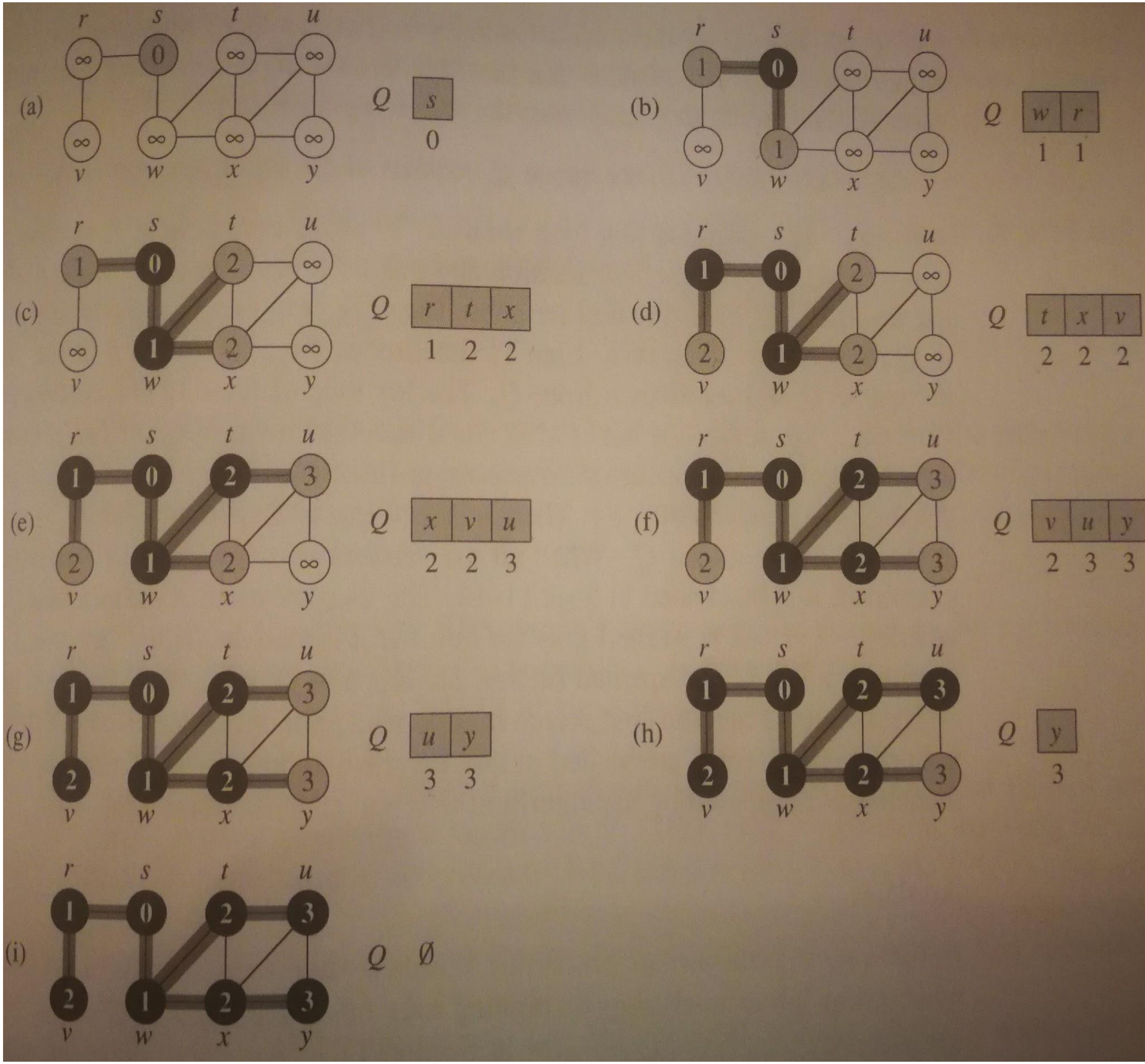


It is not visited.

when a vertex is put in, it is not finished yet.

colors to mark its status; d[u] to label its order

when it is out, it is finished.



(a) Q s 0
(b) Q w r 1 1
(c) Q r t x 1 2 2
(d) Q t x v 2 2 2
(e) Q x v u 2 2 3
(f) Q v u y 2 3 3
(g) Q u y 3 3
(h) Q y 3
(i) Q Ø
r s t u
v w x y

BFS(G, s)

```
for each vertex u ∈ V[G] − {s}
    do color[u] ← WHITE
       d[u] ← ∞
       π[u] ← NIL
color[s] ← GRAY
d[s] ← 0
π[s] ← NIL
Q ← Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if color[v] = WHITE
                 then color[v] ← GRAY
                      d[v] ← d[u] + 1
                      π[v] ← u
                      ENQUEUE(Q, v)
       color[u] ← BLACK
```

BFS Analysis

BFS may not reach all vertices.

Time = $O(V + E)$.

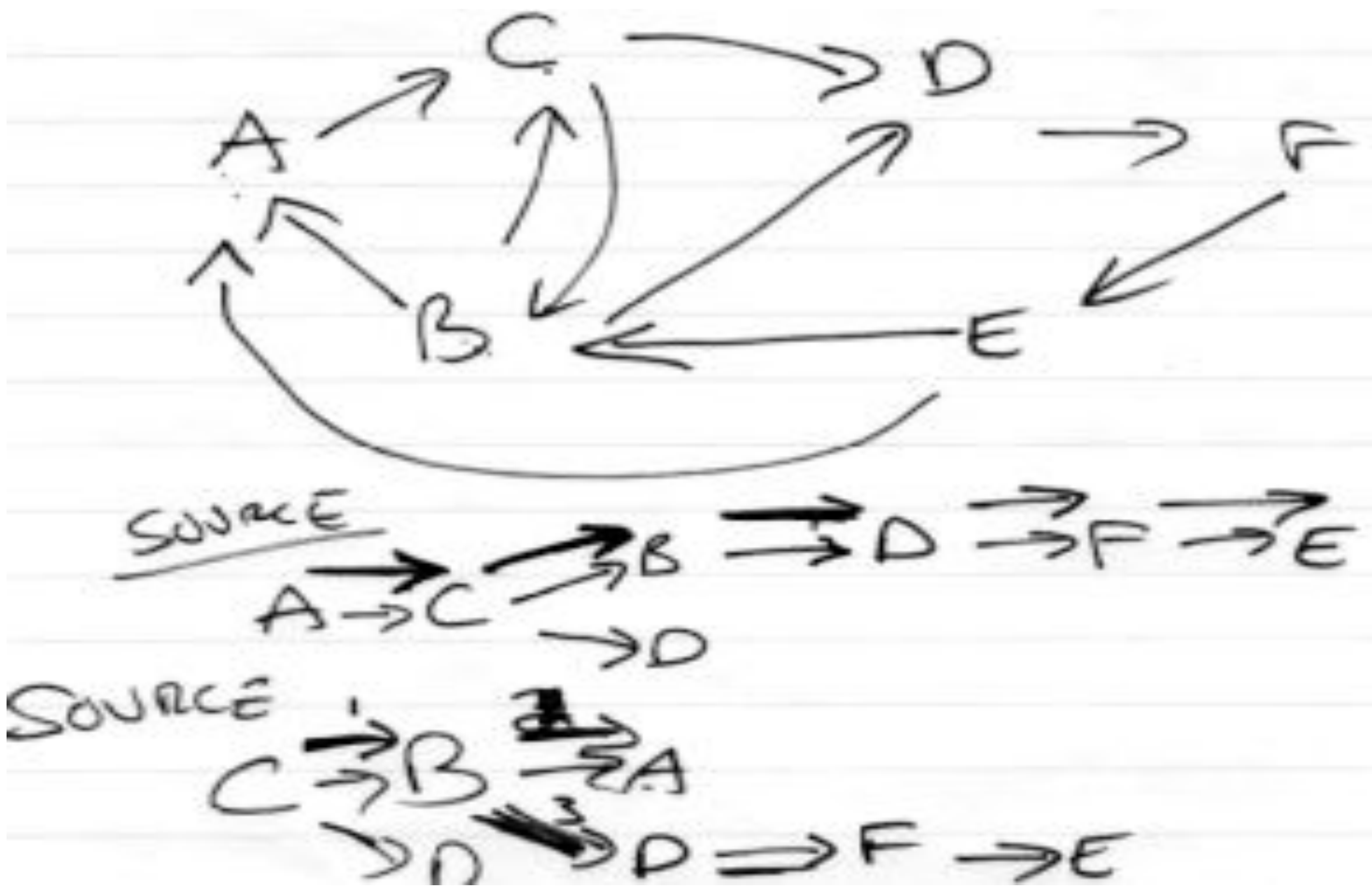
- $O(V)$ because every vertex enqueued at most once.
- $O(E)$ because every vertex dequeued at most once and we examine (u, v) only when u is dequeued. Therefore, every edge examined at most once if directed, at most twice if undirected.

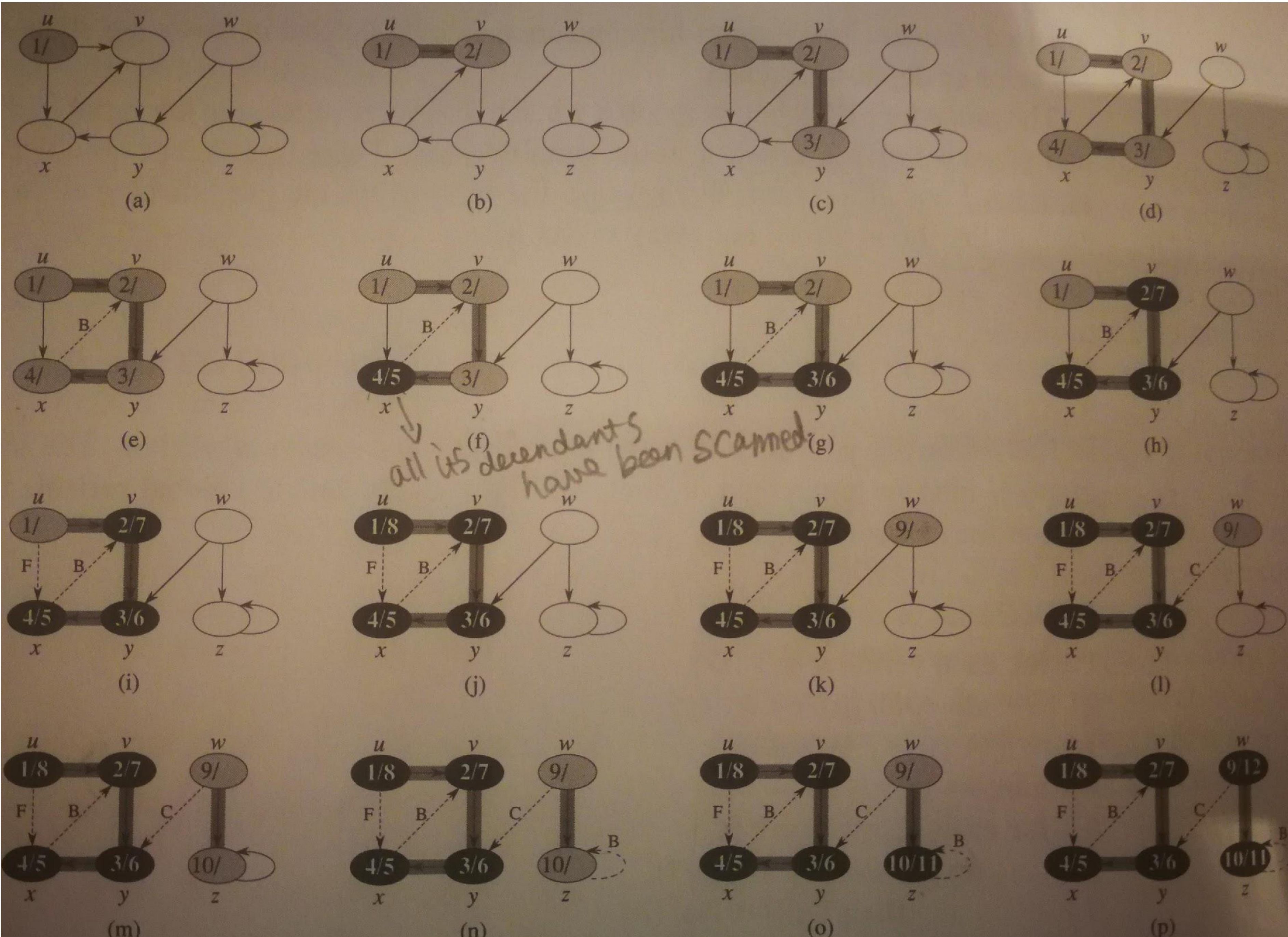
BFS and Shortest Path Lengths (minimum number of edges between 2 vertices, given source)

- BFS finds the distance (# of edges) to each reachable vertex in a graph $G = (V, E)$ from a given source vertex $s \in V$.
- Define the *shortest-path distance* $\delta(s, v)$ from s to v as the minimum number of edges in any path from vertex s to vertex v; if there is no path from s to v, then $\delta(s, v) = \infty$.
- The procedure BFS builds a **breadth-first tree** as it searches the graph

Depth First Search

As we visit a vertex we try to move to a new adjacent vertex that hasn't yet been visited, until nowhere else to go, then backtrack.





all its decendants have been scanned

```
DFS(G)
for each vertex u ∈ V[G]
     do color[u] ← WHITE
        π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
     do if color[u] = WHITE
           then DFS-VISIT(u)
```

```
DFS-VISIT(u)
color[u] ← GRAY                  ▷ White vertex u has just been discovered.
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]              ▷ Explore edge (u, v).
    do if color[v] = WHITE
          then π[v] ← u
               DFS-VISIT(v)
color[u] ← BLACK                 ▷ Blacken u; it is finished.
f[u] ← time ← time + 1
```

DFS Analysis

Time = $\Theta(V + E)$.

Similar to BFS analysis.

Θ, not just O, since guaranteed to examine every vertex and edge (by restarting from on disconnected components.

Another interesting property of depth-first search is that the search can be used to classify the edges of the input graph $G = (V, E)$.

DFS - Classification of edges

We can define four edge types in terms of the depth-first forest G_π produced by a depth-first search on G.

1. ***Tree edges*** are edges in the depth-first forest G_π. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).
2. ***Back edges*** are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. Self-loops, which may occur in directed graphs, are considered to be back edges.

DFS - Classification of edges

3. ***Forward edges*** are those nontree edges (*u*, *v*) connecting a vertex *u* to a descendant *v* in a depth-first tree.
4. ***Cross edges*** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

DFS - Classification of edges

This edge classification can be used to glean important information about a graph.

- A directed graph is acyclic if and only if a depth-first search yields no "back" edges
- In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.

Topological sort

- A depth-first search can be used to perform a topological sort of a directed acyclic graph, or a "dag" as it is sometimes called.
- A *topological sort* of a dag $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering. (If the graph is not acyclic, then no linear ordering is possible.)
- A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right. Topological sorting is thus different from the usual kind of "sorting" studied earlier.

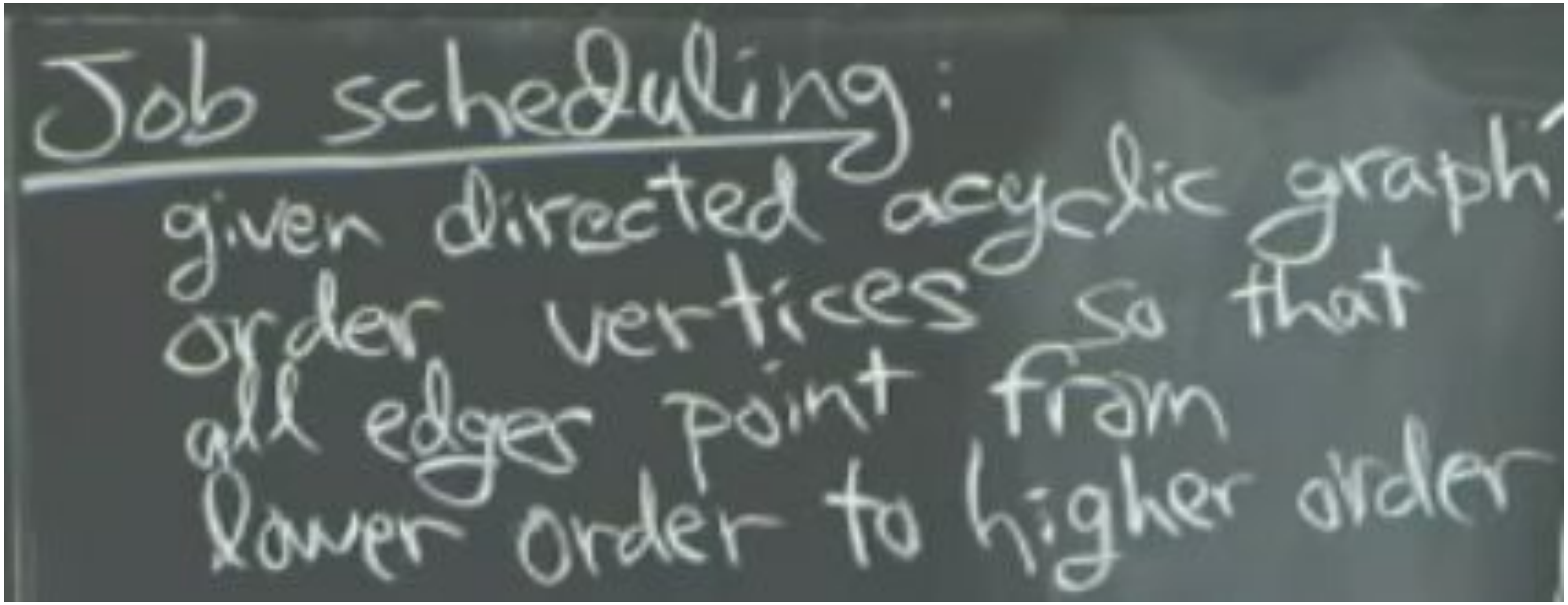
Topological sort

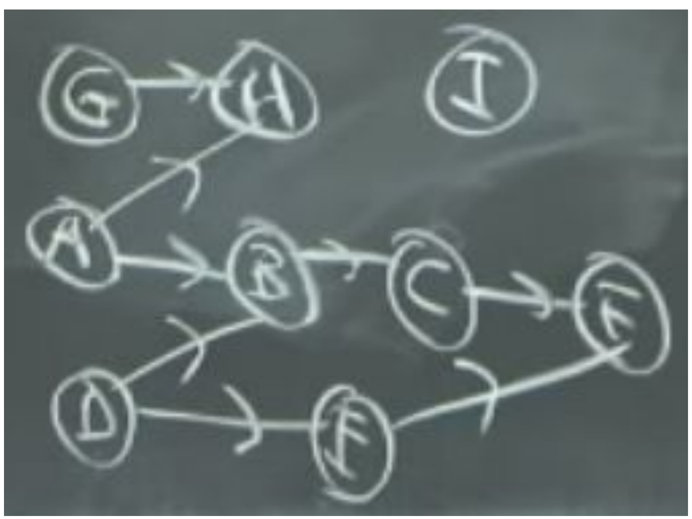
Directed acyclic graphs are used in many applications to indicate precedences among events

The following simple algorithm topologically sorts a dag.

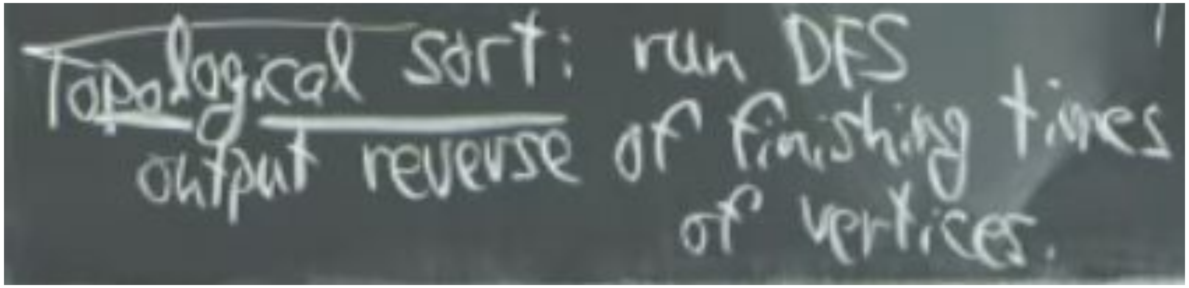
TOPOLOGICAL-SORT(G)
1 call DFS(G) to compute finishing times
$f[v]$ for each vertex v
2 as each vertex is finished, insert it onto
the front of a linked list
3 return the linked list of vertices

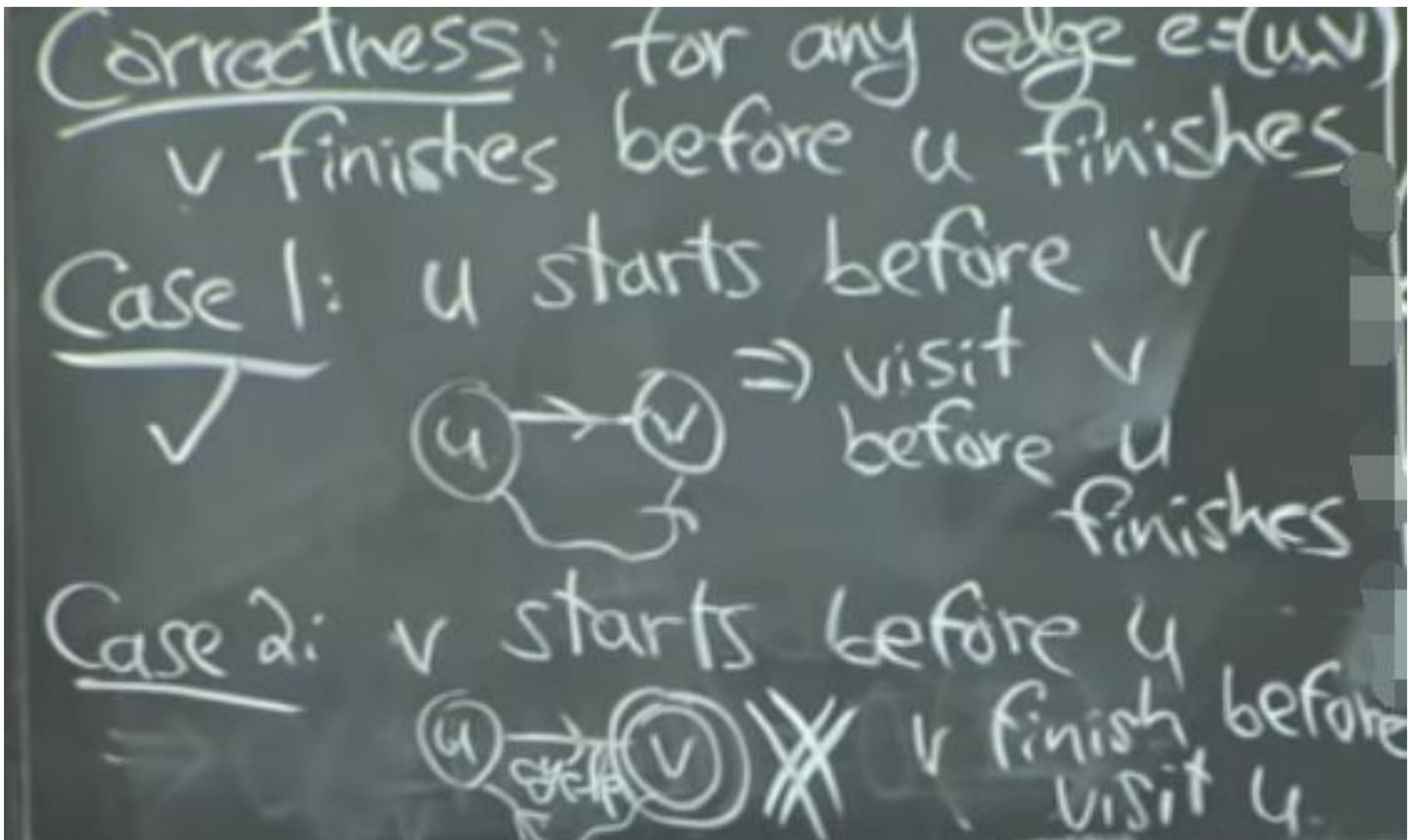
Topological sort



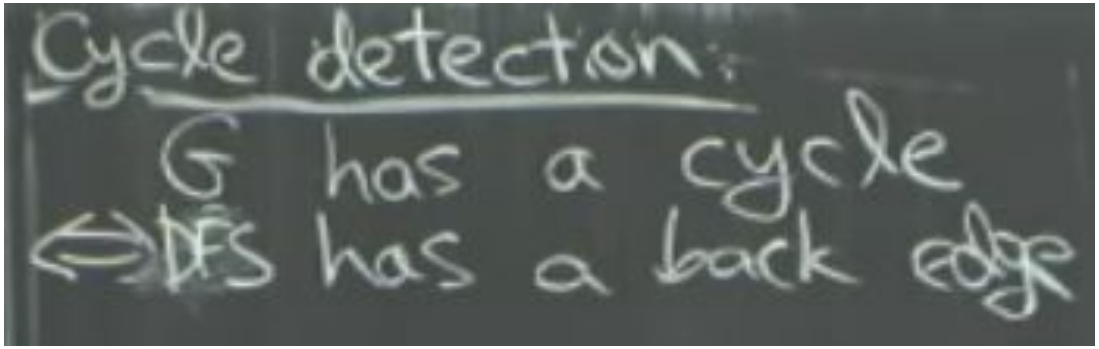


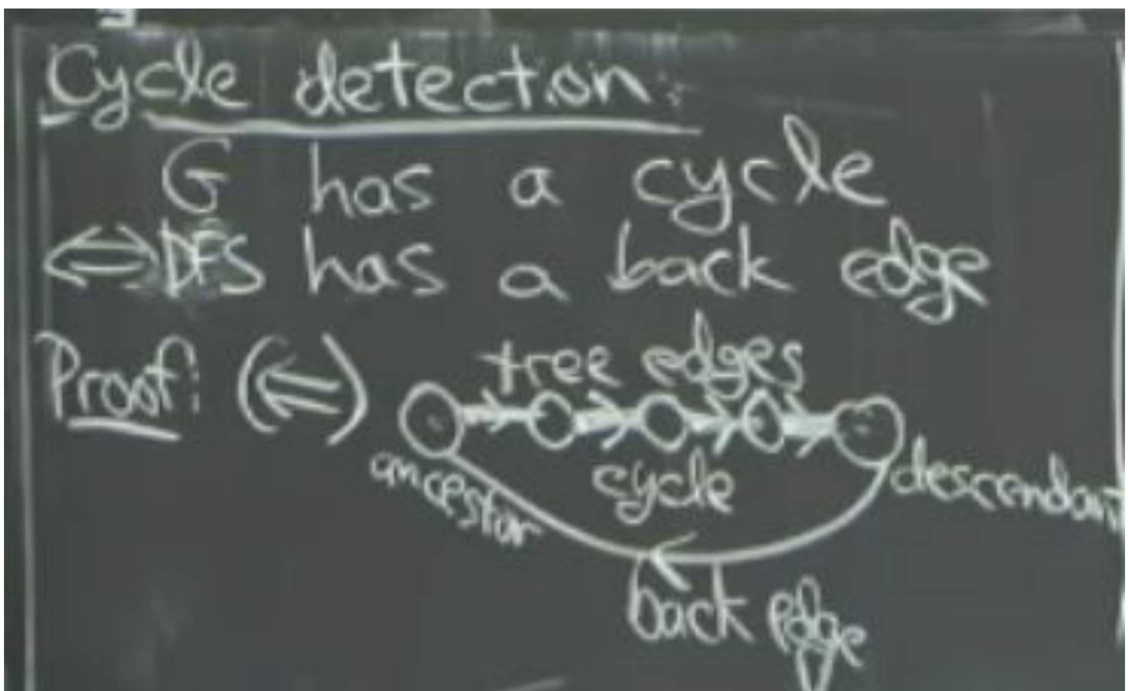
Topological sort



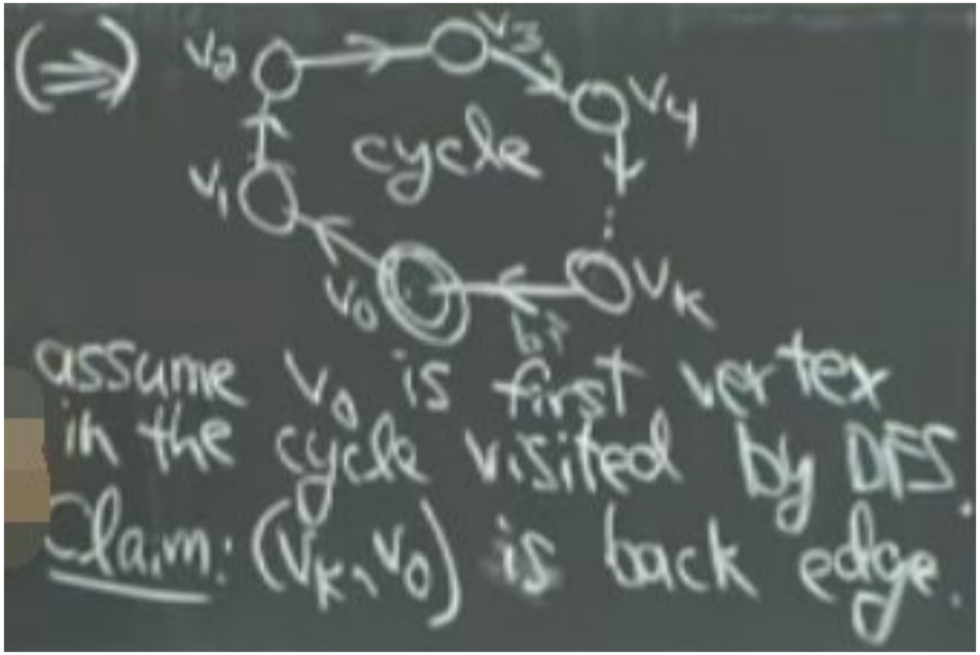


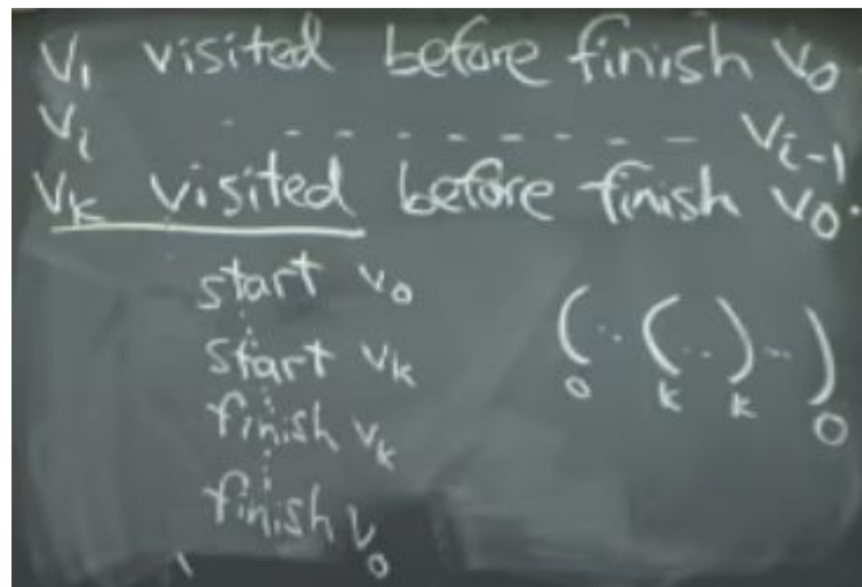
Cycle Detection





Cycle Detection





Classwork

https://student.desmos.com/?prepopulateCode=b33pqa