CSE 2320 Notes 12: Graph Representations and Search

(Last updated 10/22/06 4:40 PM)

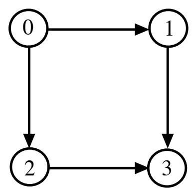
CLRS, 22.1-22.5

GRAPH REPRESENTATIONS

Adjacency Matrices – for dense $\left(E = \Omega\left(V^2\right)\right)$ and dynamic graphs

Directed Graph

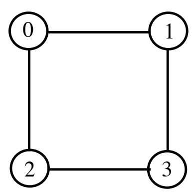
	0	1	2	3
0	0	1	1	0
1	0	0	0	1
2	0	0	0	1
3	0	0	0	0



Diagonal: Zero edges allowed for paths? (reflexive)

Undirected Graph

	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	1
3	0	1	1	0

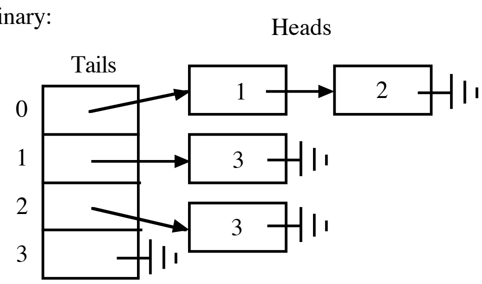


Which is more general? Time to query for presence of an edge?

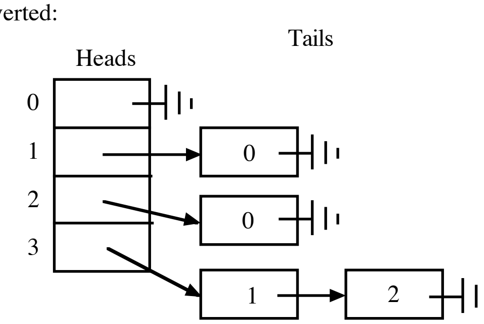
Adjacency Lists – for sparse $\left(E = O(V)\right)$ and static graphs

Directed

Ordinary:



Inverted:

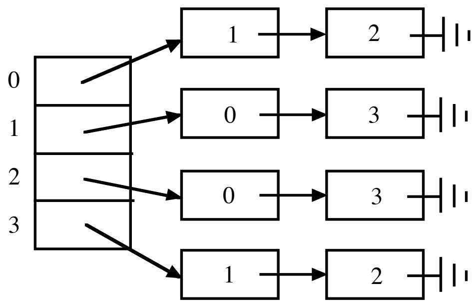


1. Time to query for presence of an edge?

2. Can convert between ordinary and inverted in $\Theta(V + E)$ time, assuming unordered lists.

3. These two structures can be integrated using both tables and a common set of nodes with two linked lists through each node.

Undirected:

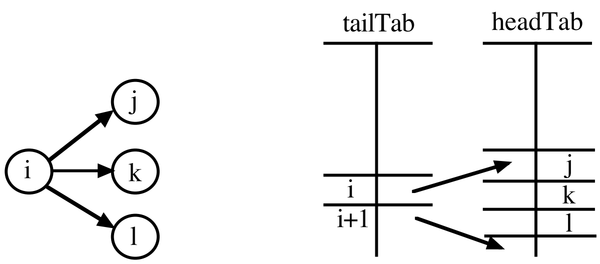


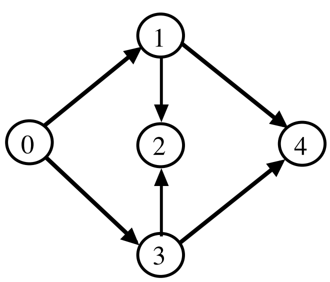
Weights – Used to represent distances, capacities, or costs.

Entries in adjacency matrix.

Field in nodes of adjacency list.

Compressed Adjacency Lists – useful for sparse, static graphs



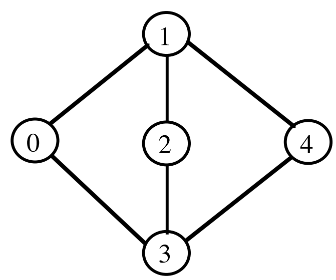


tailTab

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

headTab

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



tailTab

0	0
1	2
2	5
3	7
4	10
5	12

headTab

0	1
1	3
2	0
3	2
4	4
5	1
6	3
7	0
8	2
9	4
10	1
11	3

```
for (tail=0; tail<V; tail++)
     for (i=tailTab[tail]; i<tailTab[tail+1]; i++)
          < Process edge tail → headTab[i] >
```

Time to query for presence of an edge?

BREADTH-FIRST SEARCH (Traversal) – Queue-Based

1. Input is connected, undirected graph

   Source vertex is designated (assume 0)

   Vertex colors and interpretations

   a. White – undiscovered

   b. Gray – presently in queue

   c. Black – completely processed (all adjacent vertices have been discovered)

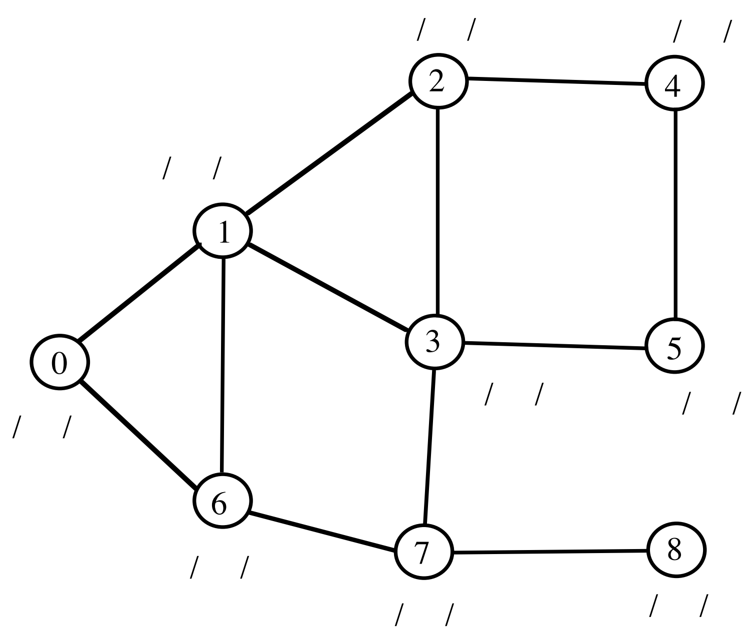
   Possible outputs:

   a. BFS number

   b. Distance (hops) from source

   c. Predecessor on BFS tree

   Label node with a/b/c



Queue:

Time:

a. Initialization (Θ(V))

b. Process each edge twice (Θ(E))

2. For disconnected, undirected graph

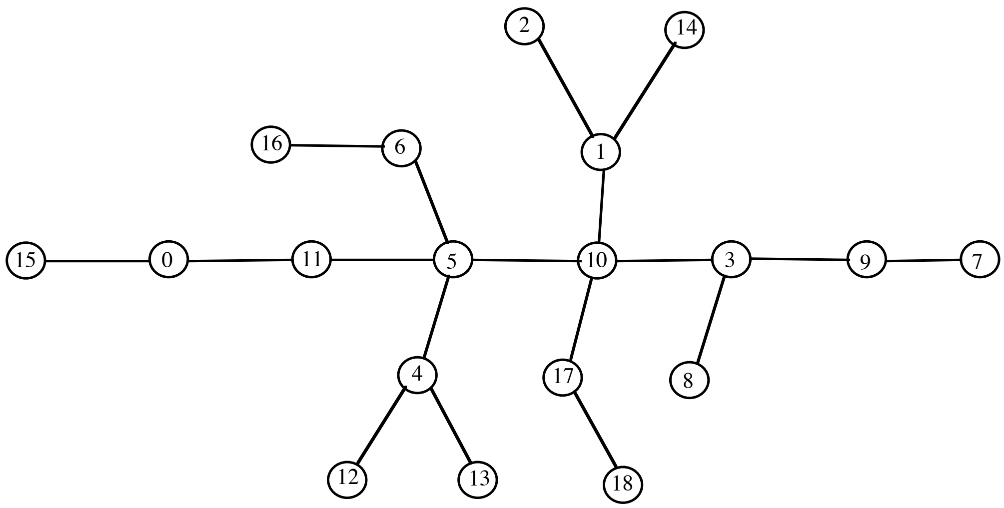
```
Initialize all vertices as white
for (i=0; i<V; i++)
        if vertex i is white
                Run BFS with i as source
```

Number of restarts is the number of components.

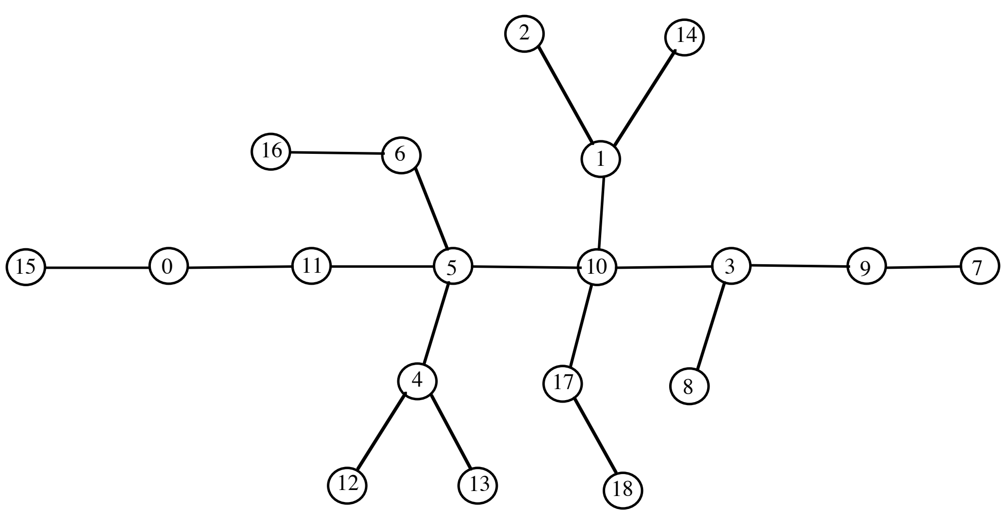
Can also use on directed graph.

Diameter of Tree – Application of BFS

1. Choose arbitrary source for BFS. Run BFS and select any vertex X at maximum distance ("hops") from source.



2. Run second BFS using X as source. X will be at one end of a diameter and any vertex at maximum distance from X can be the other end of the diameter.



Takes $\Theta(V + E)$ time.

DEPTH-FIRST SEARCH (Traversal) – Stack/Recursion-Based

Usually applied to a directed graph.

Vertex colors and interpretations

a. White – undiscovered

b. Gray – presently in stack

c. Black – completely processed (all adjacent vertices have been discovered)

Possible outputs:

a. Discovery time

b. Finish time

c. Predecessor on DFS tree

d. Edge types

Processing:

a. Change vertex from white → gray the first time it enters stack and assign discovery time (using counter).

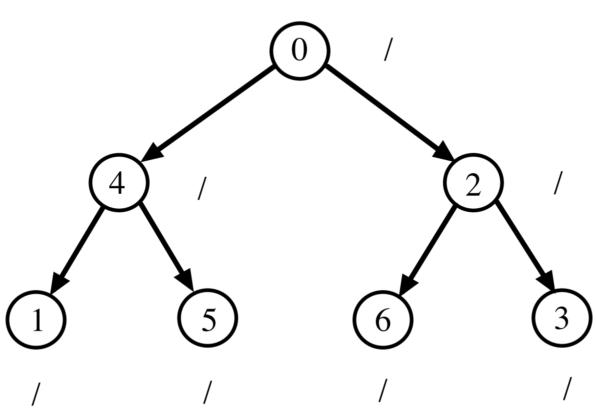
b. When a vertex (and pointer to its adjacency list) is popped, check for next adjacent vertex and push this vertex again.

c. If no remaining adjacent vertices, then change vertex from gray → black and assign finish time.

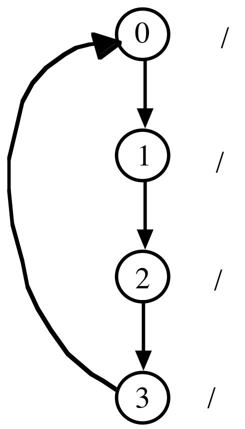
Like BFS, DFS takes Θ(V + E) time.

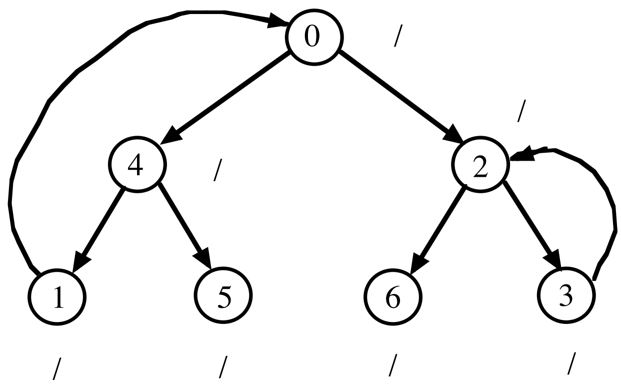
Relationship between vertex and adjacent vertex determines the *edge type*.

a. Unvisited (white) ⇒ *tree edge*



b. On the stack (gray indicating ancestor) ⇒ *back edge*

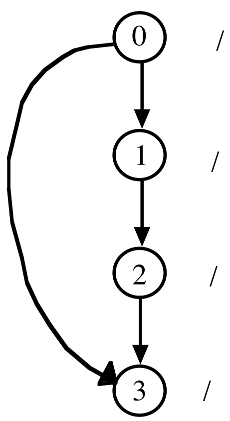


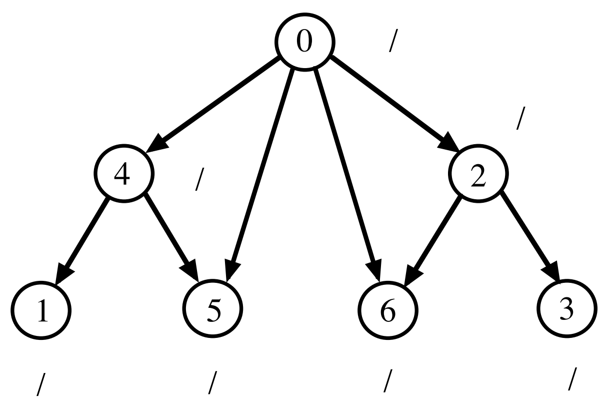


c. Previously visited, not on stack (black), but known to be descendant ⇒ *forward edge*

1. Find path of tree edges? TEDIOUS

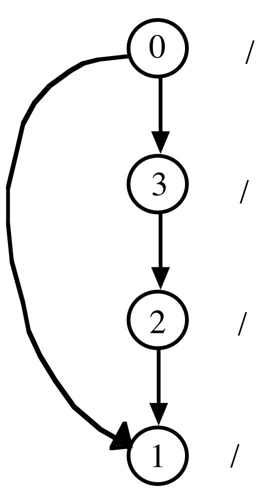
2. discovery(tail) < discovery(head)

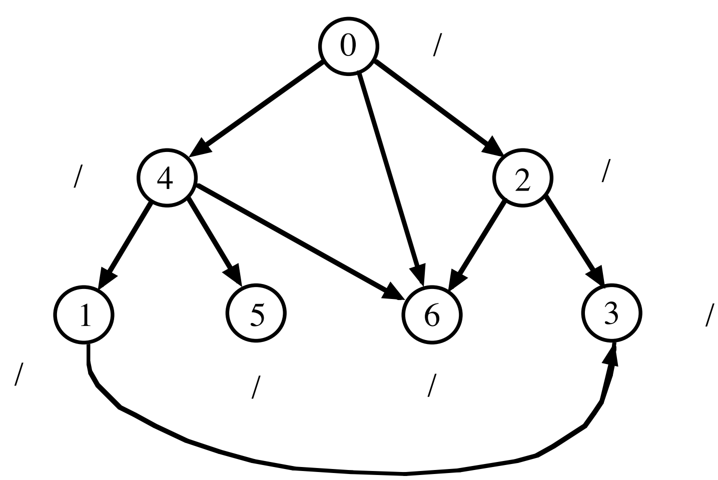




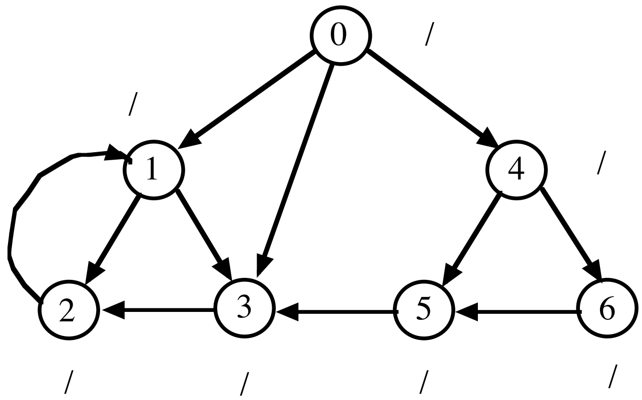
d. None of the above . . . Not on stack (black) and not a descendant ⇒ *cross edge*

Test using discovery(tail) > discovery(head)

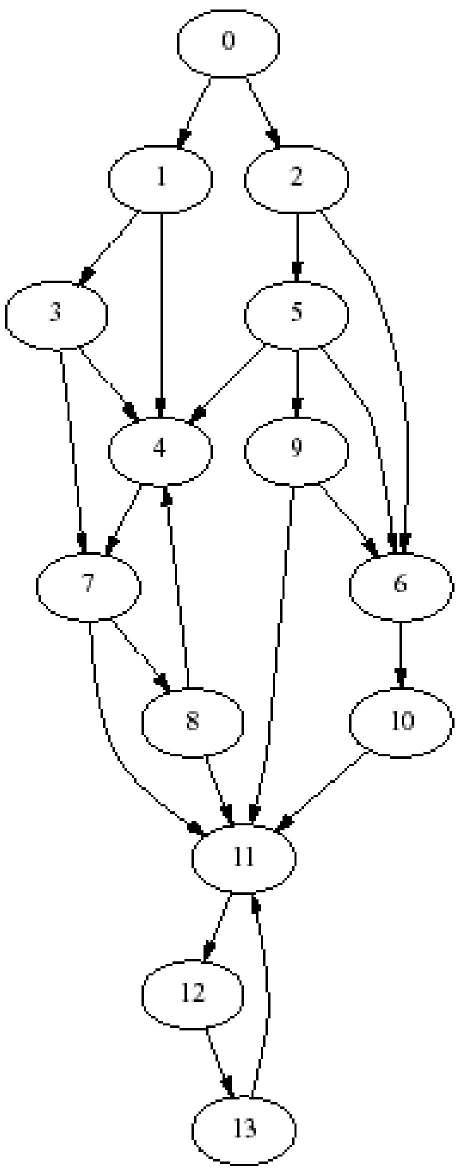




Example:



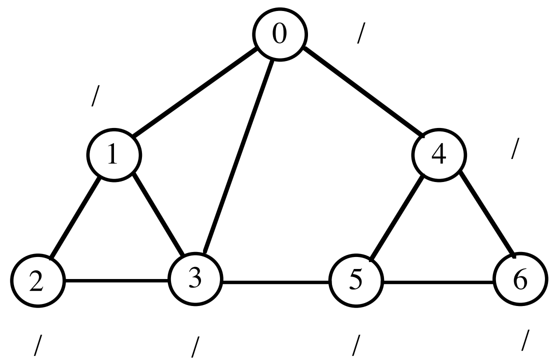
Example – available from course web page



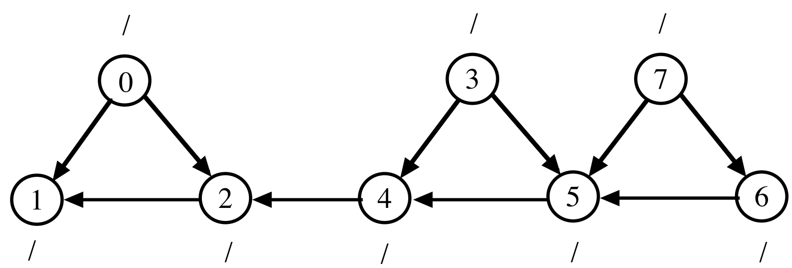
Vertex	discovery	finish	predecessor
0	1	28	-1
1	2	17	0
2	18	27	0
3	3	16	1
4	4	15	3
5	19	26	2
6	20	23	5
7	5	14	4
8	6	13	7
9	24	25	5
10	21	22	6
11	7	12	8
12	8	11	11
13	9	10	12

Edge	Tail	Head	Type
0	0	1	tree
1	0	2	tree
2	1	3	tree
3	1	4	forward
4	2	5	tree
5	2	6	forward
6	3	4	tree
7	3	7	forward
8	4	7	tree
9	5	4	cross
10	5	6	tree
11	5	9	tree
12	6	10	tree
13	7	8	tree
14	7	11	forward
15	8	4	back
16	8	11	tree
17	9	6	cross
18	9	11	cross
19	10	11	cross
20	11	12	tree
21	12	13	tree
22	13	11	back

Undirected – Can't have cross or forward edges:



Restarts – handled like BFS



TOPOLOGICAL SORT OF A DIRECTED GRAPH

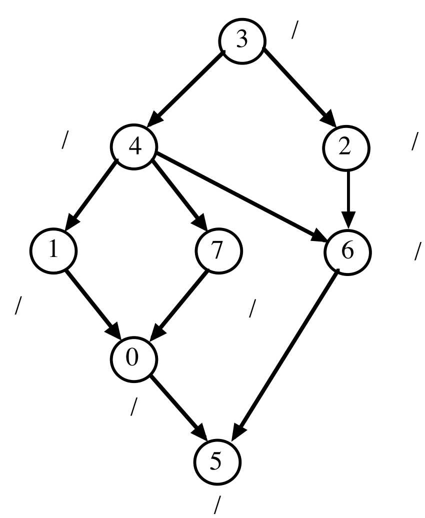
Linear ordering of all vertices in a graph.

Vertex x precedes y in ordering if there is a path from x to y in graph.

Apply DFS:

1. Back edge ⇔ graph has a cycle (no topological ordering).

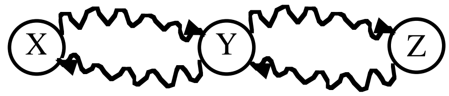
2. When vertex turns black, insert at beginning of ordering (ordering is reverse of finish times).

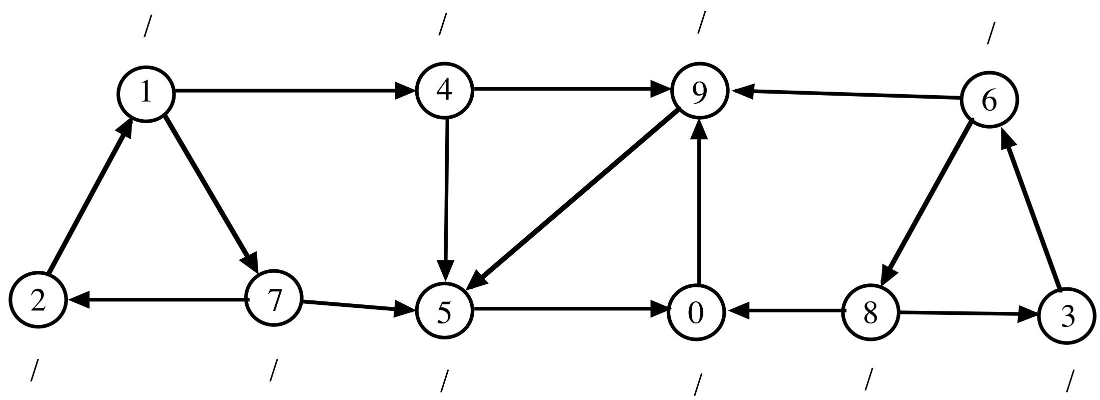


3 4 7 2 6 1 0 5

STRONGLY CONNECTED COMPONENTS

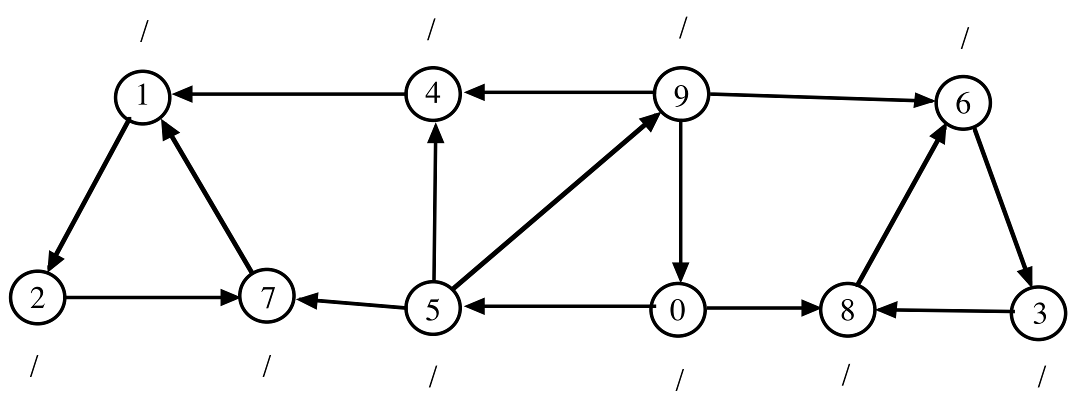
Equivalence Relation – definition (reflexive, symmetric, transitive)





1. Perform DFS. When vertex turns black ⇒ insert at beginning of list. (3 6 8 1 7 2 4 0 9 5)

2. Reverse edges.



3. Perform DFS, but each restart chooses the first white vertex in list from 1. Vertices discovered within the same restart are in the same strong component.

Observation: If there is a path from x to y and no path from y to x, then finish(x) > finish(y) (first DFS).

This implies that the reverse edge (y, x) corresponding to an original edge (x, y) without a "return path" will be a cross edge during 2nd DFS. The head vertex y will be in a SCC that has already been output.

Takes $\Theta(V + E)$ time.