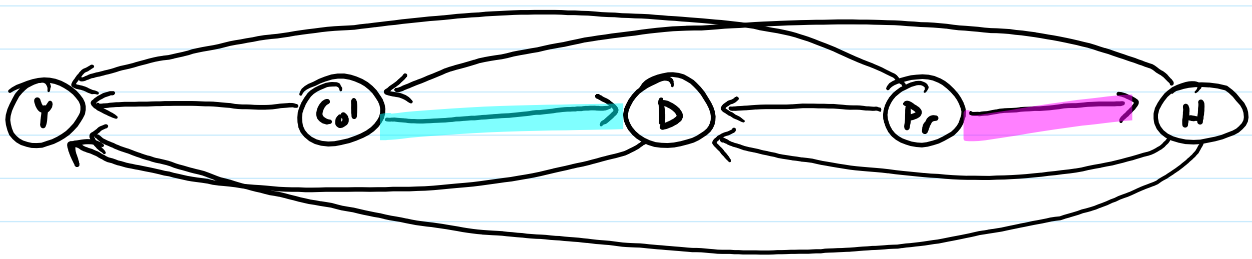
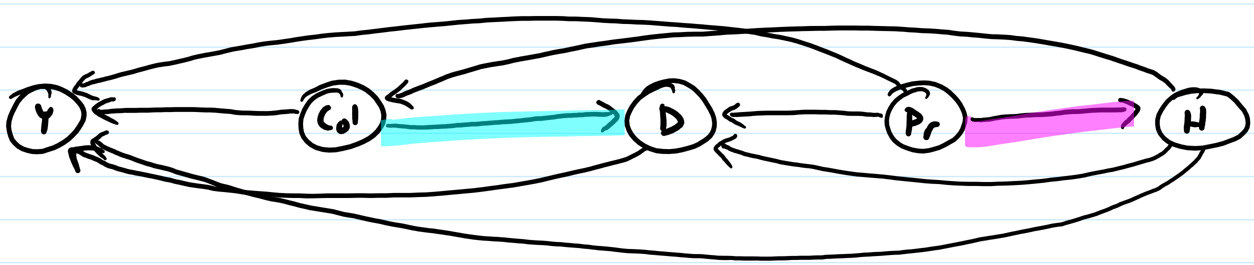
Graph Representation



Adjacency Matrix

to

		~~Y~~ 0	~~Col~~ 1	~~D~~ 2	~~Pr~~ 3	~~H~~ 4
	~~Y~~ 0	F	F	F	F	F
	~~Col~~ 1	T	F	T	F	F
from	~~D~~ 2	T	F	F	F	F
	~~Pr~~ 3	T	T	T	F	T
	~~H~~ 4	T	T	T	F	F



Adjacency List

n = 5
m = 9

~~Y~~ 0 :

~~Col~~ 1 : ~~Y~~ 0 ~~D~~ 2

~~D~~ 2 : ~~Y~~ 0

~~Pr~~ 3 : ~~Y~~ 0 ~~D~~ 2 ~~H~~ 4

$\cancel{Pr}\ 3 : \cancel{Y}\ 0 \quad \cancel{D}^{2} \quad \cancel{H}^{4}$

$\cancel{H}\ 4 : \cancel{Y}\ 0 \quad \cancel{Col}^{1} \quad \cancel{D}^{2}$

Graph Implementation Time/Space Complexity

n = # vertices sparse $m \in \Theta(n)$
m = # edges dense $m \in \Theta(n^2)$

	Adj Matrix	Adj List	Adj Set (Hash)
Space	$\Theta(n^2)$	$\Theta(n+m)$ worst case $\Theta(n^2)$ (dense) best case $\Theta(n)$ (no edges or connected)	
has_edge	$\Theta(1)$	$O(n)$ $O(\text{outdegree}(v))$	$\Theta(1)$ expected
add_edge (pre: edge doesn't exist) ↳ w/o pre, must call has_edge)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$ expected (even w/o precond)
for_each_neighbor (assuming $\Theta(1)$ processing per neighbor)	$\Theta(n)$	$\Theta(\text{outdegree}(v))$ $O(n)$	⟹
for_each_edge	$\Theta(n^2)$	$\Theta(n+m)$	⟹

for_each_edge
↓
for_each_vertex v
  for_each_neighbor (v) ⟶ for i = 0 to n - 1
    get adj list for vert i
    for each vert on that list
      do something ← assume $\Theta(1)$

$$\sum_{i=0}^{n-1} \left(1 + \text{outdegree}(\text{vert } \# i)\right)$$

$$= \sum_{i=0}^{n-1} 1 + \sum_{i=0}^{n-1} \text{outdegree}(\text{vert } \# i) \qquad \text{(directed graph)}$$

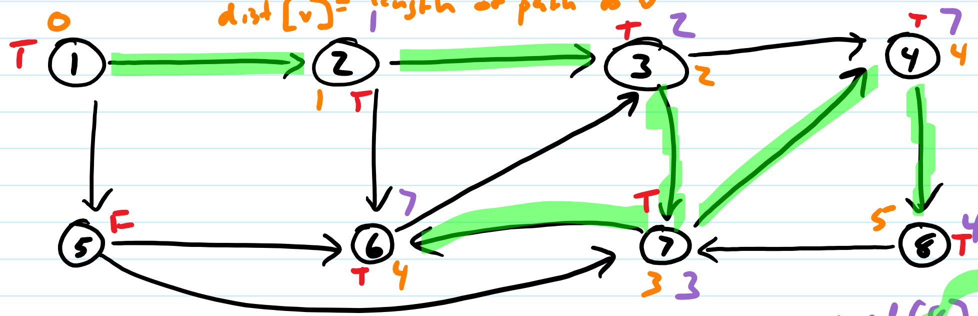
$$= n + m$$

Depth-First Search

visited[v]

pred[v] = how we got to v

dist[v] = length of path to v



pred[8] = 4
pred[4] = 7
pred[7] = 3
pred[3] = 2
pred[2] = 1

path 1 → 8

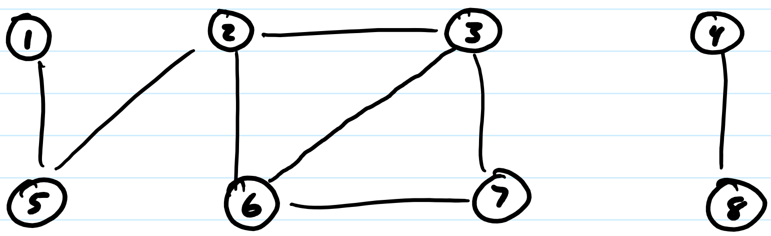
Depth-first search: Keep following edges from current vertex
Backtrack when no edges to unvisited vertices

```
DFS-VISIT(G,u)

   for each v adjacent to u
      if (color[v] = WHITE)
         pred[v] = u
         dist[v] = dist[u] + 1
         color[v] <- GRAY
         DFS-VISIT(G,v)

   color[u] <- BLACK
```

called at most once per vertex



so just like for each vertex u
for each edge (u,v)
do something

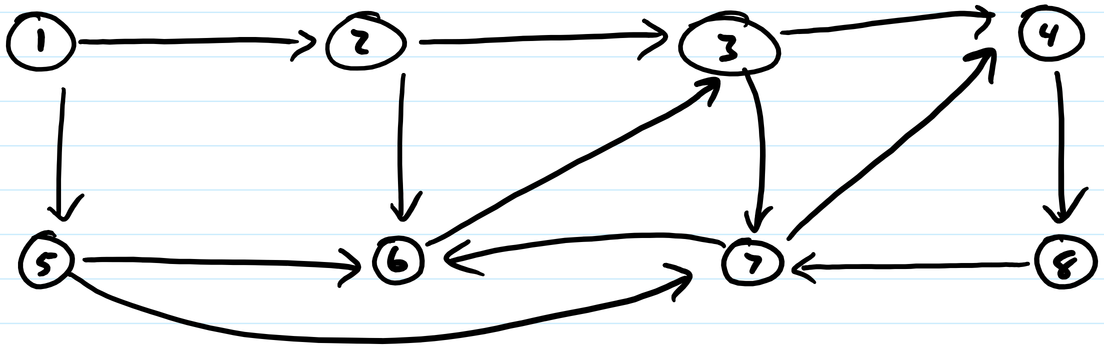
$O(n+m)$

```
DFS(G)
   for each u in G.V
      color[u] <- WHITE

   time <- 0

   for each u in G.V
      if color[u] = WHITE
         DFS-VISIT(G,u)
```

Breadth-First Search



starting from s
find verts 1 edge from s
find verts 2 edges from s
⋮

```
BFS(V,E,s)

for each vertex u in V
   color[u] <- WHITE
   d[u] <- infinity
   pred[u] <- NULL

F <- []

color[s] <- GRAY
d[s] <- 0
pred[s] <- NULL
Q <- [s]

while not Q.isEmpty()
   u <- Q.dequeue()
   for each v adjacent to u
        pred[v] <- u
        d[v] <- d[u] + 1
        color[v] <- GRAY
        Q.enqueue(v)
   color[u] = BLACK
   F = F + u
```

← iterates at most once for each vertex

$\Theta(1)$ (beside `u <- Q.dequeue()`)

so just like for each vertex u
for each edge (u,v)
do something

$\Theta(1)$ (beside `Q.enqueue(v)`)

$O(n+m)$