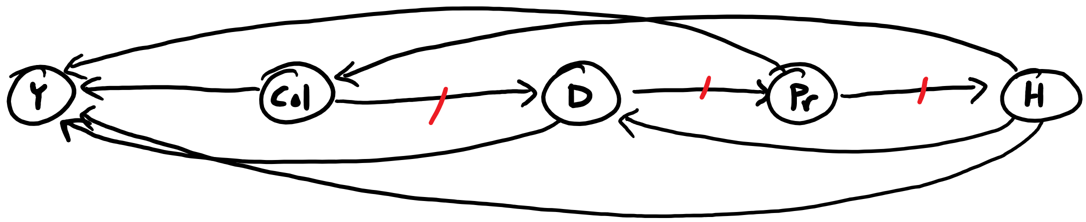
Feedback Arc Set



find ordering of vertices that minimizes wrong-way edges

vertices teams Y Col D Pr H

edges $u \to v$ means v beat u in a game

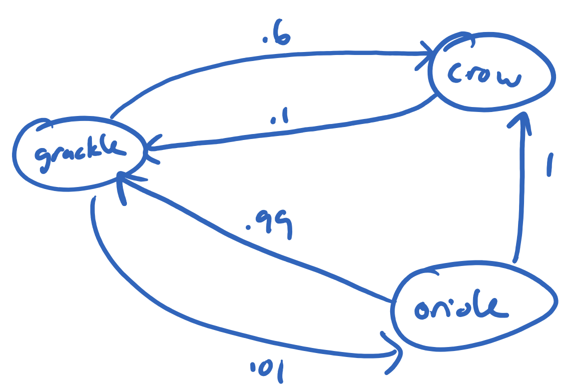
Feedback Arc Set: what is min num edges you need to remove to make graph acyclic (no cycles)

is there a cycle? [easy]

if not, find ordering so all edges go in same dir [easy]

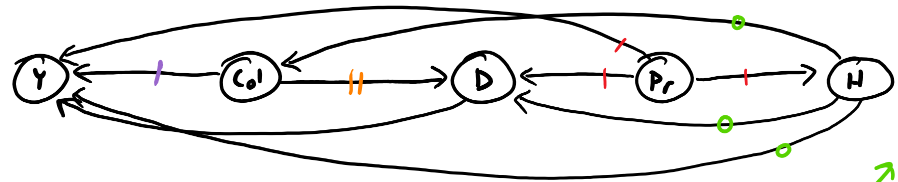
if so, find ordering to minimize # of wrong-way edges (hard: NP-complete)

brute force: for each ordering n! ordering
count # of wrong-way edges # edge
keep track of ordering giving min-so-far



what set of edges has min weight among those sets that removing them removes all cycles

Graph Representation



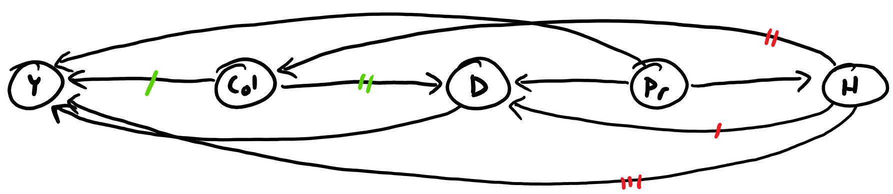
get index for from expected O(1)
get index for to
return flag at corresponding row/col O(1)

has_edge("D", "Pr") total: expected O(1)

Adjacency Matrix

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

key	values
Y	0
Col	1
D	2
Pr	3
H	4



Adjacency List (array of lists of vertices each one has an edge to)

outdegree			
0	0 Y :		
2	1 Col :	Y 0, D 2	
1	2 D :	Y 0	
3	3 Pr :	Y 0, D 2, H 4	
3	4 H :	D 2, Col 1, Y 0	
9 = m			

has_edge (from, to)
- translate from, to O(1) expected O(n) worst-case
- get list at from index O(1)
- search the list O(n) worst-case

total : O(n) worst case

Adj Set : use hash table for set representation

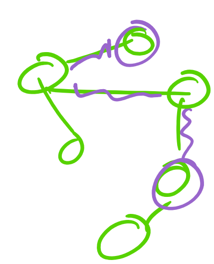
has_edge(from, to) O(1) expected

n = # vertices
m = # edges

for each edge

adj matrix : for each row r
for each column c } O(n²)
if adj[r][c] == T O(1)
process_edge(r, c)

O(n²)



if adj[r][c] == T $O(1)$

process_edge(r,c)

adj list :   for each each vertex u $O(n)$

directed graph

$\sum_u (1 + \text{outdegree}(u))$

for each vertex on adj[u]   worst case $O(n)$ $O(n^2)$ worst case total (dense)

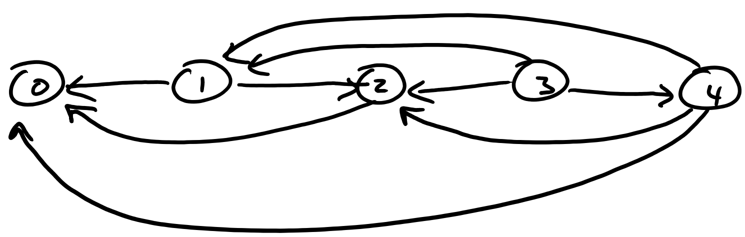
process_edge(u,v)

$O(n+m)$

$\sum_u 1 + \sum_u \text{outdegree}(u)$

$n + m$

fewest edges for undirected, connected graph = $n-1$ (sparse)



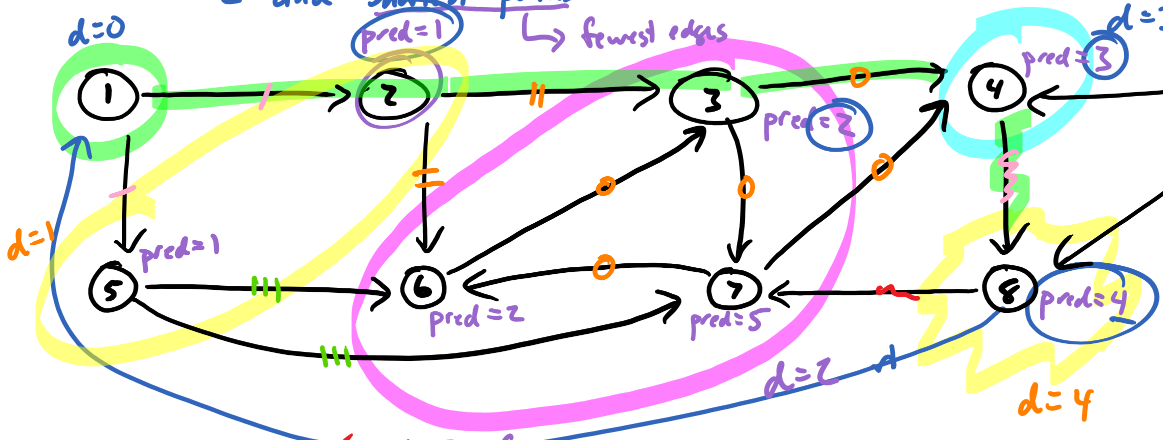
Graph Implementation Time/Space Complexity

	Adj Matrix	Adj List	Adj Set (Hash)
Space	$O(n^2)$	$O(n^2)$	$O(n+m)$
has_edge	$O(1)$	$O(n)$	$O(1)$
add_edge	$O(1)$	$O(1)$ amortized	$O(1)$
for_each_out_neighbor	$O(n)$	$O(n)$ worst case	$O(n)$
for each vertex for each_out_neighbor	$O(n^2)$	$O(n+m)$	$O(n+m)$

Breadth-First Search

- to determine which verts are reachable from start
- and shortest paths from start to each reachable vert

↳ fewest edges



1 2 3 4 8

~~1~~ ~~2~~ ~~5~~ 3 6

~~1~~ ~~2~~ ~~5~~ ~~3~~ ~~6~~ ~~7~~ ~~4~~ ~~8~~

d=0 d=1 d=2 d=3 d=4

~~1~~ ~~2~~ 5 3

```
for each vertex u                          O(n)
    color[u] = unseen

Q ← [start]                                }
color[start] ← in queue                    }
d[start] ← 0                               } O(1)
pred[start] ← NULL                         }
while Q not empty                          O(1) → for each u (in order of ↑ d)
    u ← dequeue(Q)                         O(1)
    for each outneighbor v of u                   for each out neighbor of u
        if color[v] == unseen (in queue or done)  O(1)
            enqueue(Q, v)                  O(1)
            color[v] ← in queue            O(1)
            pred[v] ← u                    O(1)
            d[v] ← d[u] + 1                O(1)
    color[u] ← done                        O(1)
```

looking at edge (u,v)

O(n+m) adj list

O(n²) adj matrix

Depth-First Search

<u>**DFS-VISIT(u)**</u>
mark u as processing

for each neighbors v of the current vertex u
if v still unseen then DFS-VISIT(v)

mark u as finished