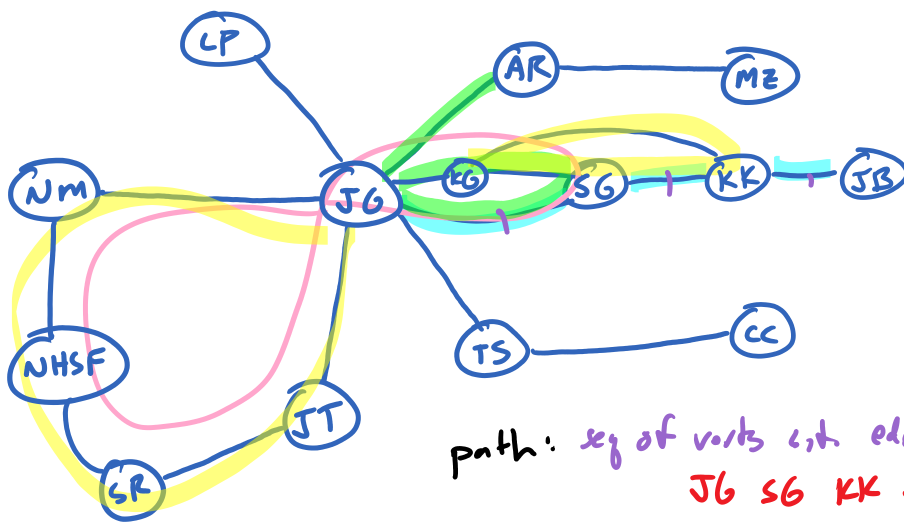
Graphs

↪ representation of things and relationships between them
vertices — edges
people — relationships

undirected



~~JG JG~~

path: seq of verts s.t. edge exists between adj verts
JG SG KK JB

simple graph: no self-loops
at most one edge per pair of verts
(per direction if a directed graph)

simple path: no repeated vertices
not simple
JG SG JG SG KK JB

cycle: path w/ same start/end
JG SG KG JG

simple cycle: only repeat is start/end

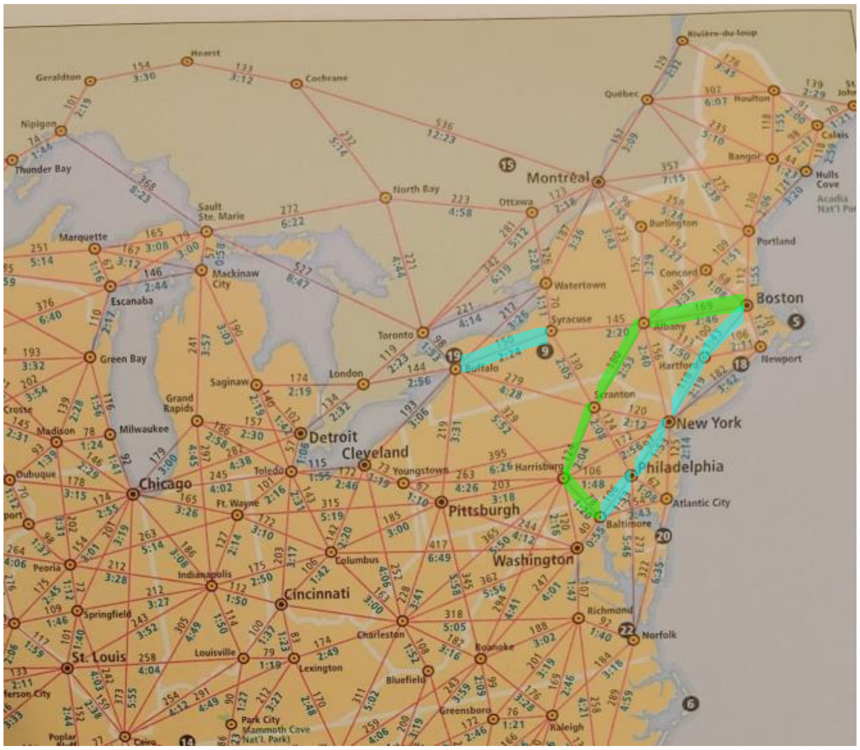
path:

simple path:

cycle:

simple cycle:

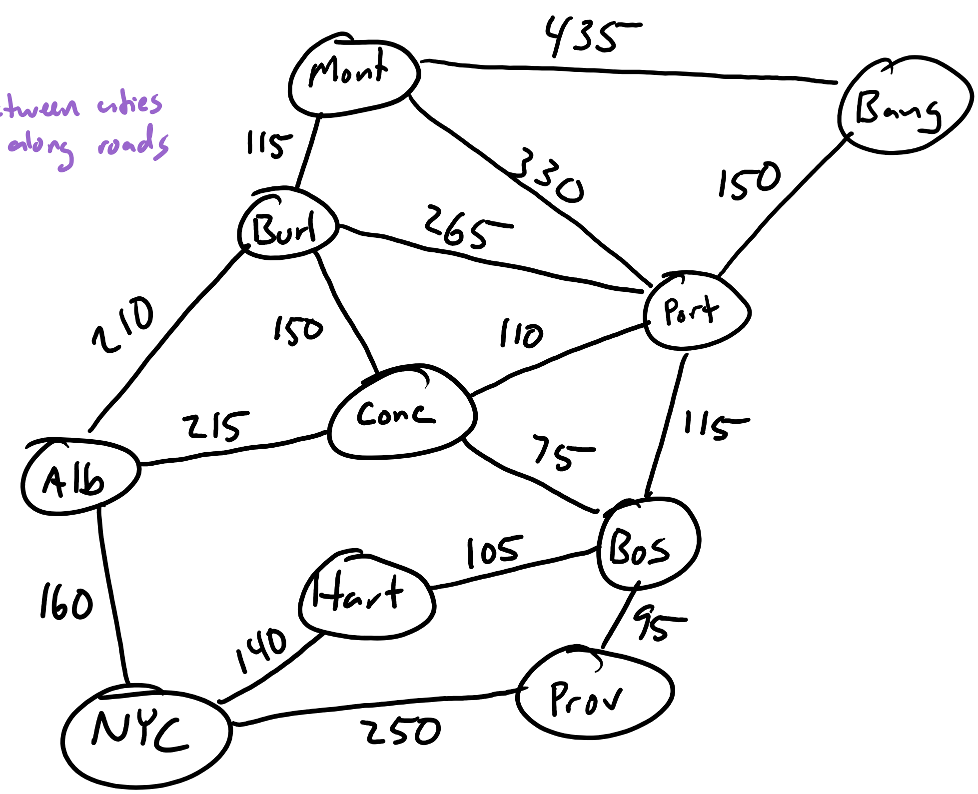
Weighted Graph

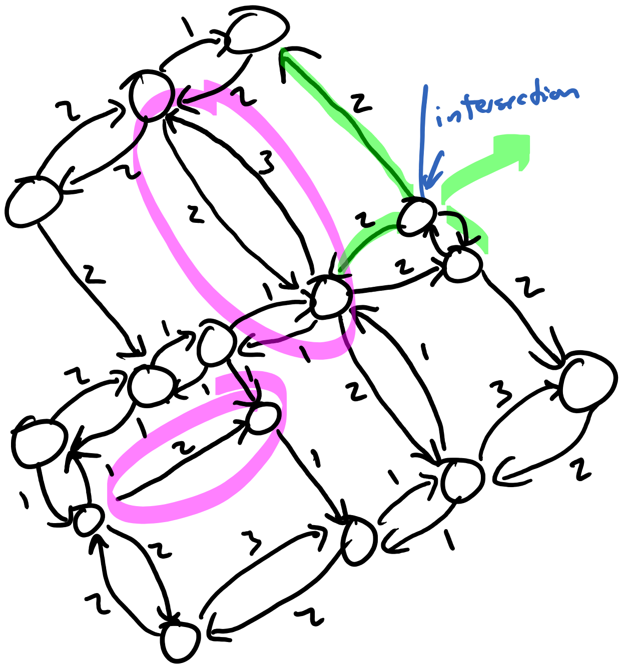


Source: Rand McNally 2012 Road Atlas

each edge labelled with weight

vertex : cities
edges : roads between cities
weights : distance along roads




intersection

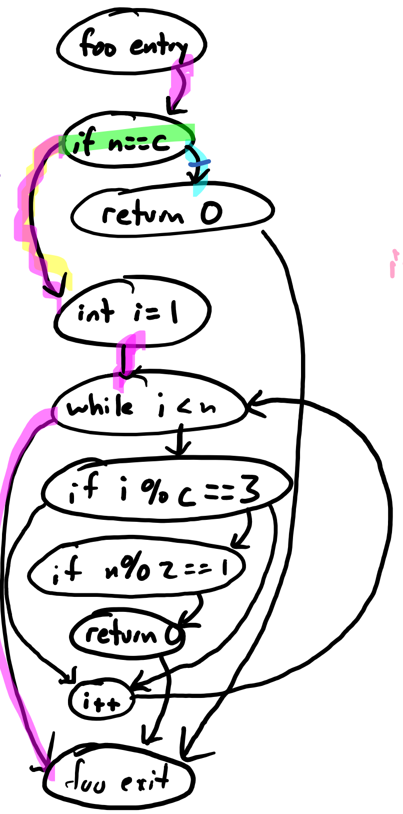

Post Office
Park
The Thirsty Pig
Takeout
Milk St
Silver St
Fire on Fore
Blyth & Burrows
Takeout
Gilberts Chowder House
Solo Italiano
Takeout
Gritty McDuff's
Brew Pub
Takeout • Delivery
Harbor Fish Market

Flow Control Graph

```
int foo(int n, int c)
{
    if (n == c)
    {
        return 0;
    }
    int i = 1;
    while (i < n)
    {
        if (i % c == 3)
        {
            if (n % 2 == 1)
            {
                return 0;
            }
        }
        i++;
    }
}
```

vertices: lines of code
edges: control flow

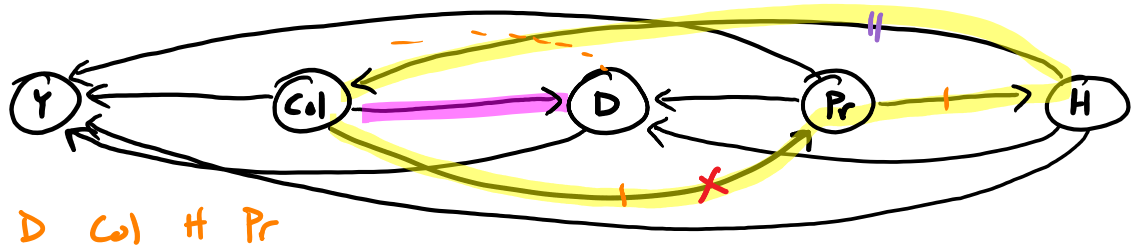
u → v
if v might immediately follow u in some execution



while (x != 0)
x++;

is there a path entry ⇝ exit w/ no return statement?

Feedback Arc Set



Y D Col H Pr

H > Pr > Col

vertices teams

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)

NP-complete (like TSP)

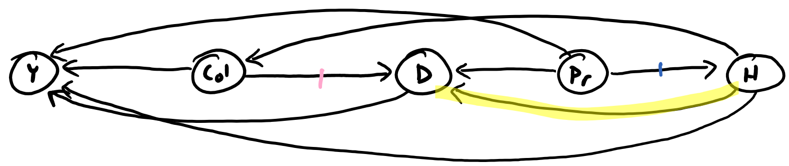
brute force: for each possible ordering $n!$
count wrong-way edges
keep track of min wrong-way

is there a cycle?

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

if so, find ordering to minimize # of wrong-way edges

Graph Representation



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

Adjacency Matrix

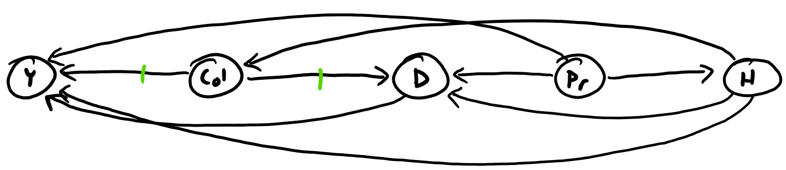
to

from		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

space: $\Theta(n^2)$

unweighted - entries are T / F

weight - entries are weights or "not there"



Adjacency List

(array of lists - one for each vert w/verts at to end of edge on list)

0 Y :

1 Col : Y^0 D^2

2 D : Y^0

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

4 H : Col^1 D^2 Y^0

n: vertices m: edges

space: worst case $\Theta(n^2)$

best case $\Theta(n)$

$\Theta(n+m)$

Adj Set: use hash table for set representation

has_edge (from, to)

adj matrix: array lookup $O(1)$

adj list: seq. search $O(n)$

add_edge (from, to)

precond: edge doesn't exist

adj matrix: array store $O(1)$

adj list: add to list $O(1)$ amortized

0:
1:
2:
3: 4
4:

for each edge

adj matrix :

```
for each row r
    for each col c
        if adj[r][c] == T
            process_edge(r, c)
```

$\Theta(n^2)$

adj list :

```
for each vertex u
    for each v in adj[u]
        process_edge(u, v)
```

$\Theta(n)$

worst case $\Theta(n)$

$\Theta(\text{outdegree}(u))$

dest vertex

$\Theta(n+m)$

worst case $\Theta(n^2)$

$\sum_{i=0}^{n-1} (1 + \text{outdegree}(v_i))$

$\Theta(n+m)$

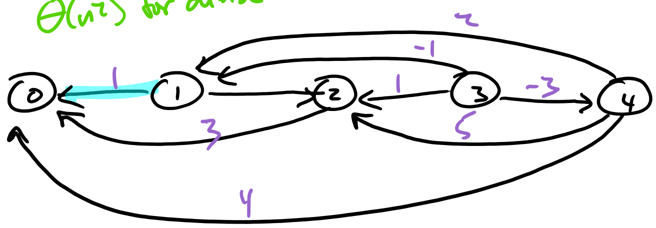
$\Theta(n)$ for sparse

$\Theta(n^2)$ for dense

for each v in adj[u]
    process_edge(u,v)

worst case $\Theta(\text{outdegree}(u))$

$$\sum_{i=0}^{n-1}(1+\text{outdegree}(v_i)) = \underbrace{\sum_{i=0}^{n-1} 1}_{n} + \underbrace{\sum_{i=0}^{n-1}\text{outdegree}(v_i)}_{m} = n + m$$



dst vertex

0:
1: {0, 1}
2:
3:
4:

weight

sparse: $m \in \Theta(n)$

dense: $m \in \Theta(n^2)$

Graph Implementation Time/Space Complexity

	Adj Matrix	Adj List	Adj Set (Hash)
Space	$\Theta(n^2)$	$\Theta(n+m)$	$\Theta(n+m)$
has_edge(from, to)	$O(1)$	$O(n)$	$O(1)$ expected
add_edge	$O(1)$	$O(1)$	$O(1)$ expected
for_each_out_neighbor	$O(n)$	$O(n)$ worst	$O(n)$ worst
for each edge for each vertex for each_out_neighbor	$\Theta(n^2)$	$\Theta(n+m)$	$\Theta(n+m)$